I. Piksele

"Podróż tysiąca mil rozpoczyna się pierwszym krokiem. " -Lao-tzu

W tej sekcji :

- Określimy współrzędne pikseli.

- Podstawowe kształty: punkt, linia, prostokąt, elipsa.

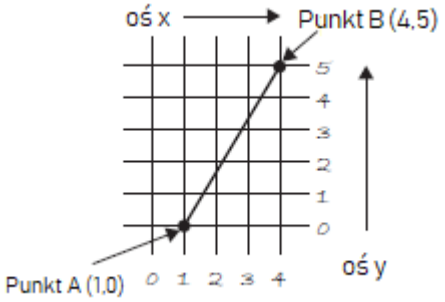
- Kolor: skala szarości, "RGB. "

- Przejrzystość kolorów.

Zauważ, że nie robimy jeszcze żadnego programowania w tym rozdziale! Po prostu czujemy się komfortowo z pomysłem tworzenia grafik na ekranie za pomocą poleceń tekstowych, czyli "kodu"!

1.1 Papier milimetrowy

Nauczysz się, jak programować w kontekście mediów obliczeniowych, i wykorzystać środowiska programistycznego Processing (http://www.processing.org) jako podstawę wszystkich dyskusji i przykładów. Ale zanim to wszystko stanie się istotne lub interesujące, musimy najpierw wyciągnąć kawałek papieru milimetrowego i narysować linię. Najkrótsza odległość między dwoma punktami to dobra staromodna linia, i tu zaczynamy, z dwoma punktami na tym papierze milimetrowym.

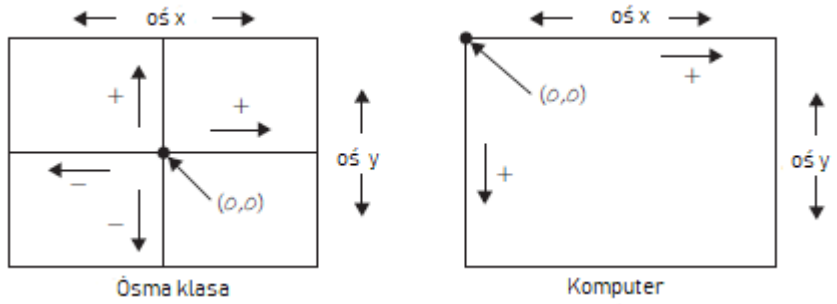


Powyższy rysunek pokazuje linię między punktem A (1,0) a punktem B (4,5). Jeśli chciałbyś poprosić swojego przyjaciela, by narysował tę samą linię powiedziałbyś "narysuj linię od punktu jeden-zero do punktu cztery-pięć, proszę." Cóż, na razie wyobraź sobie, że twój przyjaciel był komputerem i chciałeś polecić temu cyfrowemu kumplowi wyświetlenie tej samej linii na ekranie. Stosuje się to samo polecenie (tylko tym razem można pominąć uprzejmości i konieczne będzie zastosowanie precyzyjnego formatowania). Tutaj instrukcja będzie wyglądać następująco:

line (1,0,4,5);

Gratulacje, napisałeś pierwszą linię kodu komputerowego! Dojdziemy do dokładnego sformułowania powyższego później, ale na razie, nawet nie wiedząc zbyt wiele, powinno to mieć sens. Dostarczamy polecenie (które określamy mianem "funkcji"), aby maszyna nadążała za "linią". "Ponadto, podajemy kilka argumentów za tym, jak należy narysować tę linię od punktu A (0,1) do punktu B (4,5). Jeśli myślisz o tym wierszu kodu jako zdaniu, funkcja jest czasownikiem, a argumenty są obiektami zdania. Kod semantyczny również kończy się średnikiem zamiast kropką . Kluczem tutaj jest uświadomienie sobie, że ekran komputera jest niczym więcej niż bardziej wyszukanym kawałkiem papieru milimetrowego. Każdy piksel ekranu jest współrzędną - dwie liczby, "x" (poziome) i "y" (pionowe) - określające położenie punktu w przestrzeni. Naszym zadaniem jest określić, jakie kształty i kolory powinny pojawiać się na tych współrzędnych pikseli. Niemniej jednak jest tu haczyk. Układ współrzędnych

kartezjańskich umieścił (0,0) pośrodku z osią Y skierowaną do góry i osią x skierowaną w prawo (w kierunku dodatnim , kierunek ujemny w dół i w lewo). Układ współrzędnych pikseli w oknie komputera jest jednak odwracany wzdłuż osi y. (0,0) można znaleźć w lewym górnym rogu, z dodatnim kierunkiem w prawo, poziomo i pionowo w dół. Zobacz poniższy rysunek



Ćwiczenie 1-1: Patrząc na to, jak napisaliśmy instrukcję dla linii "line (1,0,4,5); " , jak myślisz jak napiszesz instrukcję rysowania Prostokąta? Koła? Trójkąta? Zapisz instrukcje w języku angielskim, a następnie przetłumaczyć je na "kod.

Angielski :

Kod :

Angielski :

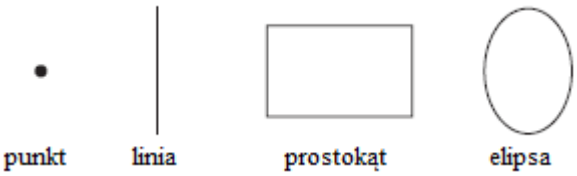
Kod :

Angielski :

Kod :

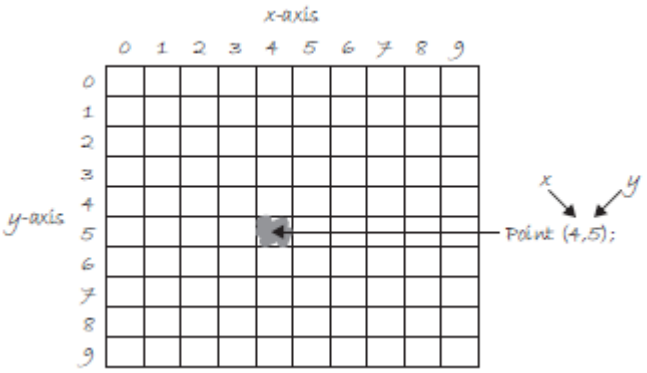
1.2 Proste Kształty

Zdecydowana większość przykładów programowania będzie miała charakter wizualny. Możesz ostatecznie nauczyć się tworzyć interaktywne gry, algorytmiczne dzieła sztuki, animowane projekty logo i (wstaw tutaj swoją własną kategorię) do przetwarzania, ale w jego centrum każdy program wizualny będzie obejmował ustawianie pikseli. Najprostszym sposobem, aby zacząć rozumieć, jak to działa, jest nauczenie się rysowania podstawowych kształtów. To nie jest niepodobne do tego jak uczymy się rysować w szkole podstawowej, tylko tutaj robimy to za pomocą kodu zamiast kredek. Zacznijmy od czterech podstawowych kształtów



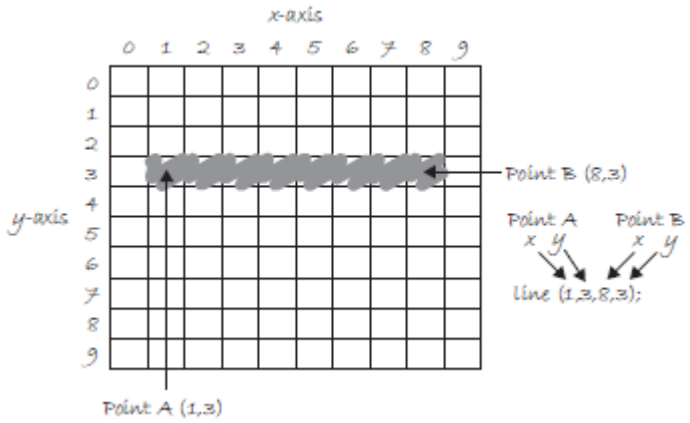
Dla każdego kształtu zadamy sobie pytanie, jakie informacje są wymagane, aby określić położenie i rozmiar (i późniejszy kolor) tego kształtu oraz dowiedzieć się, w jaki sposób przetwarzanie oczekuje na otrzymanie tych informacji. Na każdym z diagramów poniżej (rysunki od 1.5 do 1.11) zakładamy okno o szerokości 10 pikseli i wysokości 10 pikseli. To nie jest szczególnie realistyczne, ponieważ kiedy

naprawdę zaczniemy kodowanie, najprawdopodobniej będziemy pracować z dużo większymi oknami (10 x 10 pikseli to zaledwie kilka milimetrów miejsca na ekranie). Niemniej jednak dla celów demonstracyjnych dobrze jest pracować z mniejszymi liczbami, aby przedstawić piksele, które mogą pojawić się na papierze milimetrowym (na razie), aby lepiej zilustrować wewnętrzne działanie każdego wiersza kodu



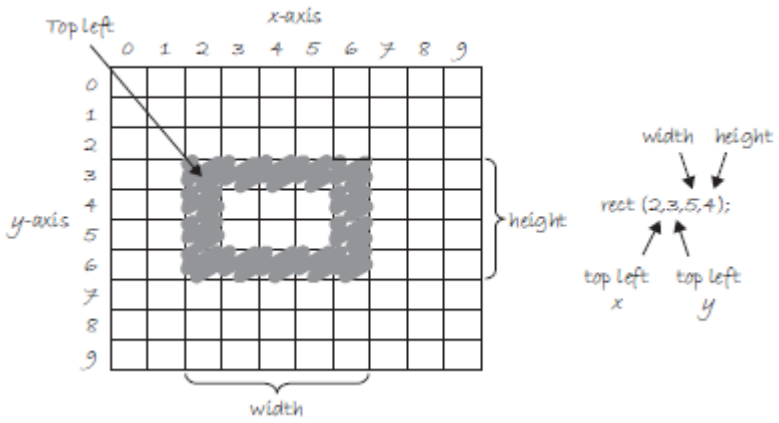
(1.5)

Punkt jest najłatwiejszy z kształtów i dobre miejsce do rozpoczęcia. Aby narysować punkt, potrzebujemy tylko współrzędnej x i y, jak pokazano na rysunku 1.5. Linia nie jest też trudna. Linia wymaga dwóch punktów, jak pokazano na rysunku 1.6.



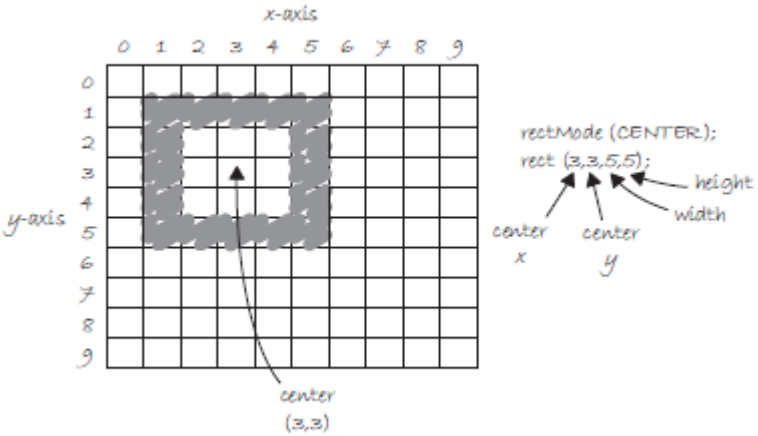
(1.6)

Gdy dojdziemy do rysowania prostokąta, sprawy stają się nieco bardziej skomplikowane. W przetwarzaniu prostokąt jest określony przez współrzędne lewego górnego rogu prostokąta, a także jego szerokość i wysokość



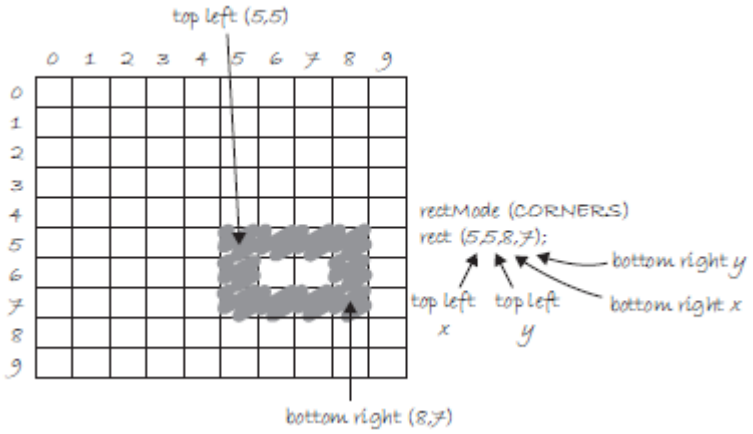
(1.7)

Jednak drugi sposób narysowania prostokąta obejmuje określenie punktu środkowego wraz z szerokością i wysokością, jak pokazano na rysunku 1.8.



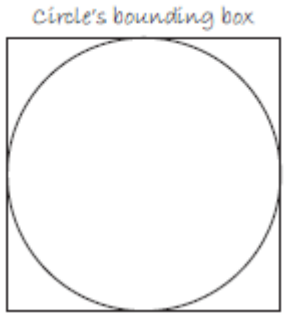
(1.8)

Jeśli wolimy tę metodę, najpierw zaznaczamy, że chcemy użyć trybu "CENTER" przed instrukcją dla samego prostokąta. Zauważ, że w przetwarzaniu rozróżniana jest wielkość liter. Nawiasem mówiąc, domyślny tryb to "NAROŻNIK", czyli jak zaczęliśmy, jak pokazano na wcześniejszym rysunku powyżej. Na koniec możemy również narysować prostokąt z dwoma punktami (lewy górny róg i dolny prawy róg). Tryb tutaj to "NARZĘDZIA"



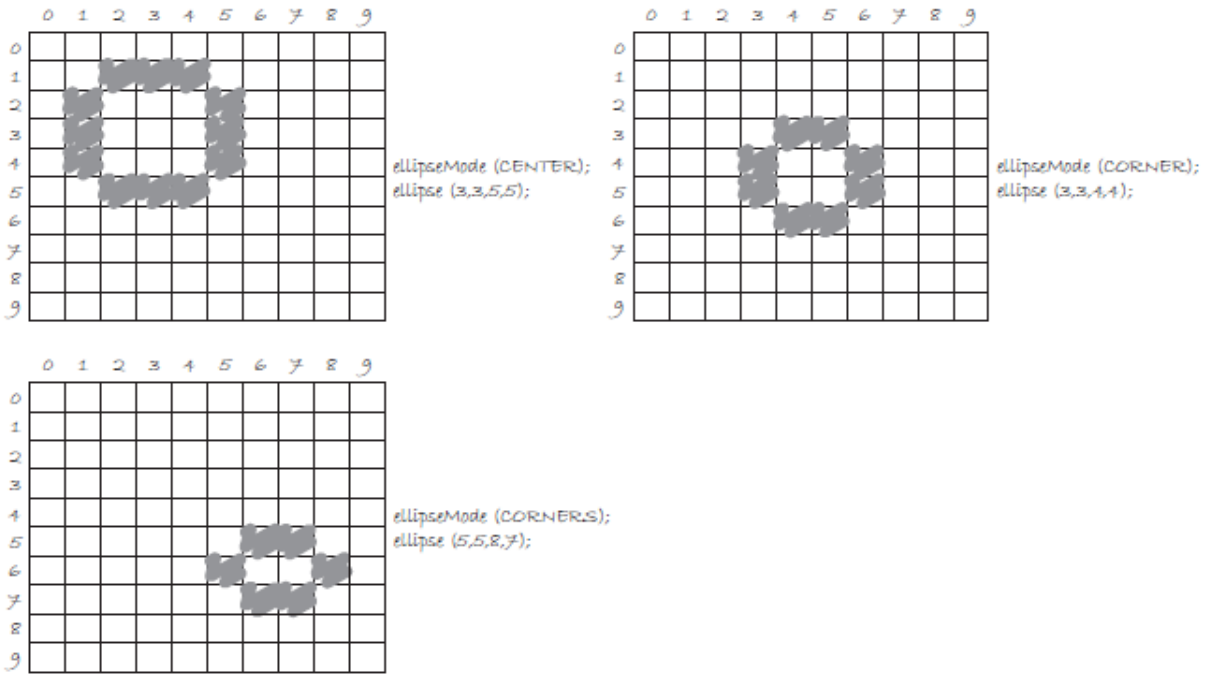
(1.9)

Kiedy już zrozumiemy koncepcję rysowania prostokąta, elipsa jest bardzo prosta. W rzeczywistości jest to identyczne z rect (), z tą różnicą, że rysowana jest elipsa, gdzie byłby prostokąt ograniczający 1 (jak pokazano na rysunku 1.11)



(1.11)

prostokąta. Domyślnym trybem dla elipsy () jest "CENTER", a nie "CORNER" jak przy rect (). Zobacz rysunek 1.10.

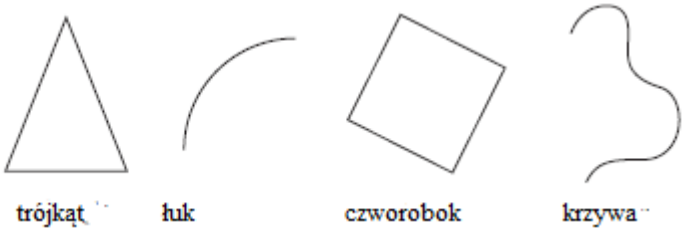


(1.10)

Ważne jest, aby przyznać, że na rysunku 1.10 elipsy nie wyglądają szczególnie okrągłe. Processsing ma wbudowaną metodologię wyboru pikseli, które mają być używane do utworzenia okrągłego kształtu. Powiększony w ten sposób, otrzymujemy kilka kwadratów w kształcie koła, ale pomniejszonych na ekranie komputera, otrzymujemy ładną okrągłą elipsę. Później zobaczymy, że Processing daje nam siłę do opracowania własnych algorytmów dla kolorowanie w poszczególnych pikselach (w rzeczywistości możemy już sobie wyobrazić, jak możemy to zrobić za pomocą "punktu" w kółko), ale na razie jesteśmy zadowoleni z umożliwienia wykonania "elipsy" ciężkiej pracy. Oczywiście, punkt, linia, elipsa i prostokąt nie są jedynymi kształtami dostępnymi w bibliotece funkcji przetwarzania. W rozdziale 2 zobaczymy,

jak odniesienie do przetwarzania dostarcza nam pełną listę dostępnych funkcji rysowania wraz z dokumentacją wymaganych argumentów, składni przykładowej i obrazów. Na razie, jako ćwiczenie, możesz spróbować wyobrazić sobie, jakie argumenty są wymagane w przypadku innych kształtów (rysunek 1.12):

triangle()
arc()
quad()
curve()



Ćwiczenie 1-2: Korzystając z pustego wykresu poniżej, narysuj pierwotne kształty określone przez kod

linia (0,0,9,6);

punkt (0,2);

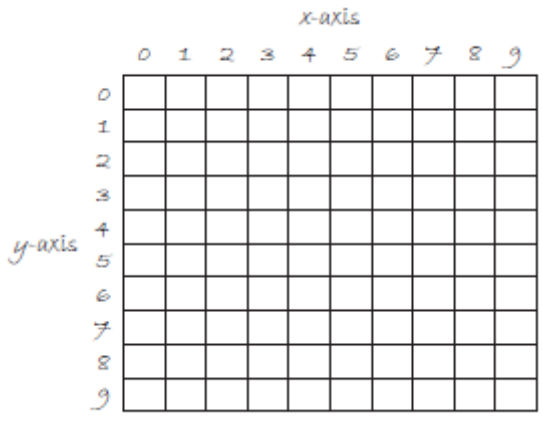
punkt (0,4);

rectMode (CORNER);

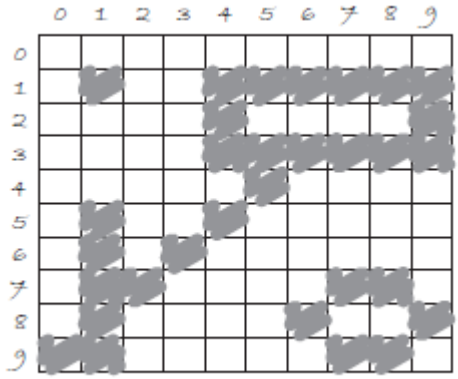
rect (5,0,4,3);

ellipseMode (CENTER);

elipsa (3,7,4,4);



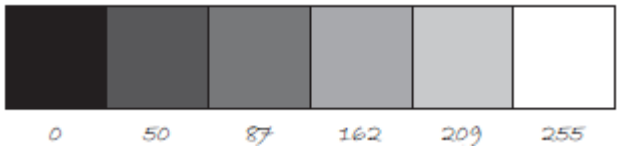
Ćwiczenie 1-3: Odwrotna inżynieria listy prostych instrukcji rysowania kształtów dla diagramu



Jest więcej niż jedno rozwiązanie

1.3 Kolor szarości

Jak dowiedzieliśmy się w sekcji 1.2, głównym elementem konstrukcyjnym do umieszczania kształtów na ekranie jest współrzędna pikselowa. Grzecznie poinstruowałeś komputer, aby narysował kształt w określonej lokalizacji o określonym rozmiarze. Brakowało jednak podstawowego elementu - koloru. W świecie cyfrowym wymagana jest precyzja. Mówiąc "Hej, możesz sprawić, że krąg będzie niebiesko-zielony? ". Dlatego kolor jest definiowany za pomocą szeregu liczb. Zacznijmy od najprostszego przypadku: czarno-białego lub skali szarości. W skali szarości mamy następujące: 0 oznacza czarny, 255 oznacza biały. W międzyczasie każdy inny numer - 50, 87, 162, 209 itd. - ma odcień szarości w zakresie od czerni do bieli.



Czy 0-255 wydaje ci się arbitralne?

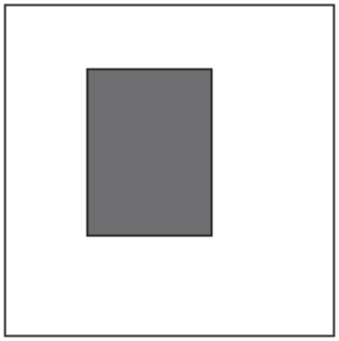
Kolor dla danego kształtu musi być przechowywany w pamięci komputera. Th to pamięć to tylko długa sekwencja zer i jedynek (cała masa przełączników włączających i wyłączających). Każdy z tych przełączników jest nieco, osiem z nich razem jest bajtem. Wyobraź sobie, że mamy osiem bitów (jeden bajt) w sekwencji - jak wiele sposobów możemy skonfigurować te przełączniki? Odpowiedź brzmi (i zrobimy małe badanie liczb binarnych, które to potwierdzą) 256 możliwości, lub zakres liczb od 0 do 255. Będziemy używać ośmiobitowego koloru dla naszego zakresu skali szarości i 24 bity dla pełnego koloru (osiem bitów dla każdego z elementów koloru czerwonego, zielonego i niebieskiego). Zrozumienie, w jaki sposób działa ten zakres, możemy teraz przejść do ustawiania specyficznych kolorów w skali szarości dla kształtów, które narysowaliśmy w sekcji 1.2. W przetwarzaniu każdy kształt ma stroke () lub fill () lub oba. stroke () jest konturem kształtu, a fill () jest wnętrzem tego kształtu. Linie i punkty mogą mieć tylko stroke(), z oczywistych powodów. Jeśli zapomnimy podać kolor, Processing będzie domyślnie używać koloru czarnego (0) dla stroke() i białego (255) dla funkcji fill (). Zwróć uwagę, że teraz używamy bardziej realistycznych liczb dla lokalizacji pikseli, zakładając większe okno o wielkości 200 200 pikseli.

rect(50,40,75,100);

Dodając funkcje stroke () i fill () przed narysowaniem kształtu, możemy ustawić kolor. To bardzo przypomina instruowanie twojego przyjaciela, aby używał pióra do rysowania na papierze milimetrowym. Musisz powiedzieć przyjacielowi, zanim zacznie rysować, a nie po. Istnieje również funkcja background (), która ustawia kolor tła okna, w którym będą renderowane kształty.

Przykład 1-1: Obrys i wypełnienie

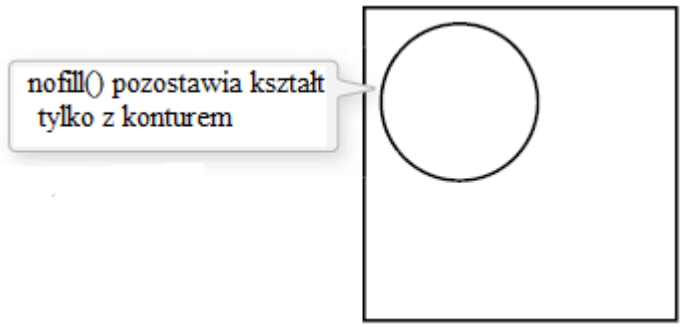
```
background(255);

stroke(0);

fill(150);

rect(50,50,75,100);
```



stroke () lub fill () można wyeliminować za pomocą funkcji noStroke () lub noFill (). Naszym instynktem może być powiedzenie "stroke (0)" bez żadnego konturu, jednak ważne jest, aby pamiętać, że 0 nie jest "nic", ale raczej oznacza kolor czarny. Pamiętaj też, aby nie eliminować obu - przy pomocy noStroke () i noFill () nic się nie pojawi!

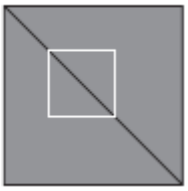
Przykład 1-2: noFill ()

```
background(255);

stroke(0);

noFill();

ellipse(60,60,100,100);
```

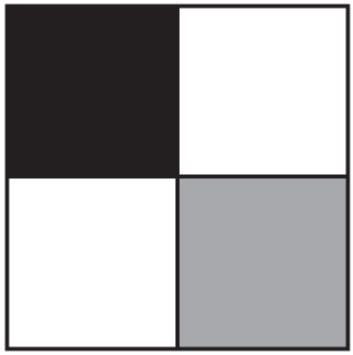


Jeśli narysujemy dwa kształty naraz, Processing zawsze użyje ostatnio określonego stroke () i fill (), odczytując kod od góry do dołu

```
background(150);
stroke(0);
line(0,0,100,100);
stroke(255);
noFill();
rect(25,25,50,50);
```



Ćwiczenie 1-4: Spróbuj zgadnąć, jakie będą instrukcje dla następującego zrzutu ekranu.



1.4 Kolor RGB

Nostalgiczne spojrzenie na papier milimetrowy pomogło nam nauczyć się podstaw lokalizacji i wielkości pikseli. Teraz, gdy nadszedł czas, aby przestudiować podstawy cyfrowego koloru, szukamy kolejnego wspomnienia z dzieciństwa, która pozwoli nam zacząć. Pamiętaj o malowaniu palców? Mieszając trzy "podstawowe" kolory, można wygenerować dowolny kolor. Wirowanie wszystkich kolorów razem spowodowało zabłocony kolor brązowy. Im więcej farby dodałeś, tym jest ciemniej. Kolory cyfrowe są również konstruowane poprzez mieszanie trzech podstawowych kolorów, ale działa to różnie w zależności od farby. Po pierwsze, różne są podstawowe: czerwony, zielony i niebieski (tj. Kolor "RGB"). A w kolorze na ekranie mieszamy światło, a nie malujemy, więc zasady mieszania również są różne.

• Czerwony + Zielony = Żółty

• Czerwony + niebieski = Fioletowy

• Zielony + Niebieski = Cyjan (niebiesko-zielony)

• Czerwony + Zielony + niebieski = biały

• Brak kolorów = czarny

Zakłada się, że kolory są tak jasne, jak to tylko możliwe, ale oczywiście masz dostępną gamę kolorów, więc niektóre czerwone plus trochę zieleni plus trochę niebieskiego są szare, a odrobina czerwieni plus odrobina niebieskiego to ciemnyfiolet . Chociaż może to trochę potrwać , im więcej programujesz i eksperymentujesz z kolorem RGB, tym bardziej stanie się on instynktowny, podobnie jak wirujące kolory z palcami. I oczywiście nie można powiedzieć "pomieszaj trochę czerwieni z odrobiną niebieskiego", musisz podać dokładną ilość. Podobnie jak w skali szarości, poszczególne elementy koloru są wyrażone w zakresie od 0 (brak tego koloru) do 255 (w miarę możliwości) i są wymienione w kolejności R, G i B. Otrzymasz zawieszkę RGB mieszanie kolorów poprzez eksperymenty, ale następnie omówimy niektóre z nich za pomocą niektórych popularnych kolorów.

Zwróć uwagę, że my pokazujemy tylko czarno-białe wersje każdego szkicu przetwarzania, ale wszystko jest udokumentowane online w pełnym kolorze na stronie http://www.learningprocessing.com z diagramami kolorów RGB, które można znaleźć na stronie: http://learningprocessing.com /color .

Przykład 1-3 . Kolory RGB

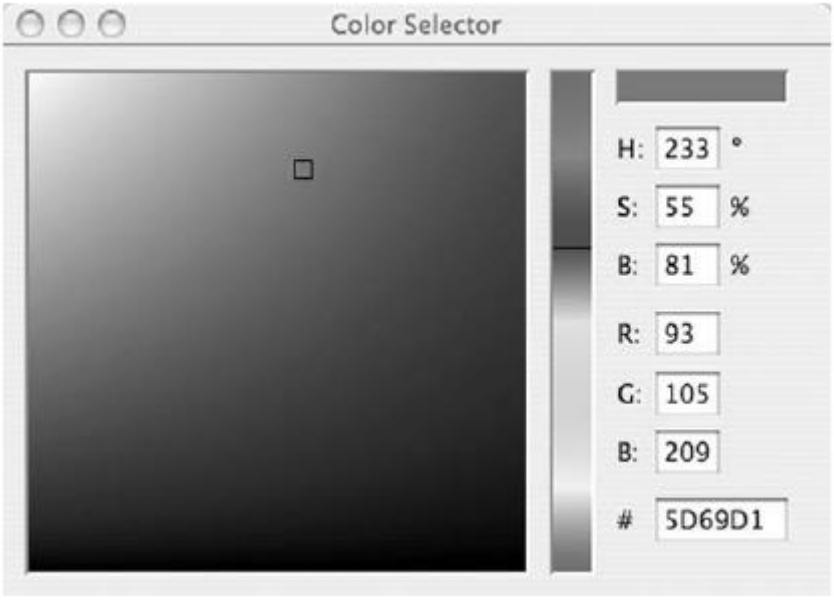
```
background(255);
noStroke();

fill(255,0,0);
ellipse(20,20,16,16);

fill(127,0,0);
ellipse(40,20,16,16);

fill(255,200,200);
ellipse(60,20,16,16);
```

Jasno czerwony

Ciemno czerwony

Różowy

Przetwarzanie ma również selektor kolorów, aby pomóc w wyborze kolorów. Dostęp do tego można uzyskać za pośrednictwem TOOLS (z paska menu) → COLOR SELECTOR.



Ćwiczenie 1-5: Wypełnij następujący program. Zgadnij, jakich wartości RGB użyć (będziesz w stanie sprawdzić swoje wyniki w Processing po przeczytaniu następnej części). Możesz także użyć selektora kolorów pokazanego powyżej

```
fill(_________,_________,_________);
ellipse(20,40,16,16);
fill(_________,_________,_________);
ellipse(40,40,16,16);
fill(_________,_________,_________);
ellipse(60,40,16,16);
```

Jasnoniebieski

Fioletowy

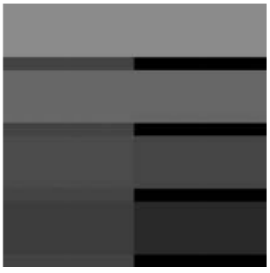
Żółty

Ćwiczenie 1-6: Jaki kolor będzie generował każdy z poniższych wierszy kodu?

```
fill(0,100,0);
fill(100);
stroke(0,0,200);
stroke(225);
stroke(255,255,0);
stroke(0,255,255);
stroke(200,50,50);
```

1.5 Przejrzystość kolorów

Oprócz czerwonych, zielonych i niebieskich składników każdego koloru, istnieje dodatkowy opcjonalny czwarty składnik, określany jako "alpha" koloru. "Alpha oznacza przezroczystość i jest szczególnie przydatna, gdy chcesz narysować elementy, które częściowo wyglądają jako jeden nad drugim. Wartości alfa dla obrazu są czasami określane zbiorczo jako "kanał alfa" obrazu. Ważne jest, aby zdać sobie sprawę, że piksele nie są dosłownie przezroczyste, jest to po prostu wygodne złudzenie, które uzyskuje się poprzez mieszanie kolorów. Za kulisami Processing przyjmuje numery kolorów i dodaje znak procenta od jednego do procentu innego, tworząc optyczną percepcję mieszania. (Jeśli jesteś zainteresowany programowaniem okularów "w kolorze różowym", to tutaj zaczynasz.) Wartości alfa również mieszczą się w zakresie od 0 do 255, przy czym 0 jest całkowicie przezroczyste (tj. 0% nieprzejrzysty) i 255 całkowicie nieprzezroczyste (tj. 100% nieprzejrzysty). Przykład 1-4 pokazuje przykład kodu pokazany na rysunku



Przykład 1-4: Przejrzystość alfa

```
background(0);
noStroke();

fill(0,0,255);
rect(0,0,100,200);

fill(255,0,0,255);
rect(0,0,200,40);

fill(255,0,0,191);
rect(0,50,200,40);

fill(255,0,0,127);
rect(0,100,200,40);

fill(255,0,0,63);
rect(0,150,200,40);
```

Brak czterech argumentów oznacza 100% nieprzeźroczystości

255 oznacza 100% nieprzeźroczystości

75% nieprzeźroczystości

50 % nieprzeźroczystości

25% nieprzeźroczystości

1.6 Niestandardowe zakresy kolorów

Kolor RGB z zakresami od 0 do 255 to nie jedyny sposób na przetwarzanie kolorów w przetwarzaniu. Za kulisami w pamięci komputera, kolor jest zawsze mówiony jako seria 24 bitów (lub 32 w przypadku kolorów z alfą). Processing pozwoli nam jednak myśleć o kolorze w dowolny sposób, i przetłumaczyć nasze wartości na liczby, które komputer rozumie. Na przykład możesz preferować kolor jako od 0 do 100 (jak procent). Możesz to zrobić, określając własny colorMode ().

colorMode (RGB, 100); : Za pomocą colorMode () możesz ustawić własny zakres kolorów.

Powyższa funkcja mówi: "OK, chcemy myśleć o kolorze pod względem czerwieni, zieleni i niebieskiego. Zakres wartości RGB będzie wynosić od 0 do 100. "Chociaż rzadko jest to wygodne, możesz także mieć różne zakresy dla każdego składnika koloru:

colorMode (RGB, 100,500,10,255);

Teraz mówimy "Czerwone wartości idą od 0 do 100, zielone od 0 do 500, niebieskie od 0 do 10, a alpha od 0 do 255." Na koniec, podczas gdy będziesz prawdopodobnie potrzebował tylko koloru RGB dla wszystkich Twoich potrzeb programistycznych, możesz także określić kolory w trybie HSB (odcień, nasycenie i jasność). Bez zbytniej szczegółowości, kolor HSB działa w następujący sposób:

• Barwa - rodzaj koloru, domyślnie wynosi od 0 do 360 (myśl o 360 ° na kolorowym "kółku").

• Nasycenie - Żywotność koloru, od 0 do 100 domyślnie.

• Jasność - Jasność, jasność koloru, domyślnie od 0 do 100.

Ćwiczenie 1-7: Zaprojektuj stworzenie używając prostych kształtów i kolorów. Narysuj stwora ręcznie, używając tylko punktów, linii, prostokątów i elips. Następnie spróbuj wpisać kod dla stworzenie za pomocą komend przetwarzania opisanych w tym rozdziale: point (), lines (), rect (), ellipse (), stroke () i fill (). W następnej części będziesz miał szansę przetestować swoje wyniki, uruchamiając swój kod w Processing.

Przykład 1-5: Stworek

```
ellipseMode(CENTER);
rectMode(CENTER);
stroke(0);
fill(150);
rect(100,100,20,100);
fill(255);
ellipse(100,70,60,60);
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
stroke(0);
line(90,150,80,160);
line(110,150,120,160);
```

Przykładową odpowiedzią jest Stworek. W ciągu dziewięciu części będziemy śledzić przebieg dzieciństwa Stworka. Podstawy programowania zostaną zademonstrowane w miarę jego rozwoju. Najpierw nauczymy się wyświetlać go, a następnie tworzyć interaktywnym i i animowanym, a wreszcie powielać go w świecie wielu Stowrków Proponuję zaprojektować własną "rzecz" (zauważ, że nie ma potrzeby ograniczania się do humanoidalnej lub podobnej do stwora formy, jakikolwiek programowy wzór) i odtworzenia wszystkich przykładów w pierwszych dziewięciu rozdziałach z własnym projektem. Najprawdopodobniej będzie to wymagać jedynie zmiany małej część (część renderująca kształt) każdego przykładu. Proces ten jednak powinien pomóc w utrwaleniu zrozumienia podstawowych elementów wymaganych w programach komputerowych - zmiennych, warunkowych, pętli, funkcji, obiektów i tablic - i przygotować się, gdy Stworek dojrzeje, opuści gniazdo i odejdzie w bardziej zaawansowane tematy.

II. Processing

2.1 Processing na ratunek

Teraz, gdy podbiliśmy świat prymitywnych kształtów i kolorów RGB, jesteśmy gotowi wdrożyć tę wiedzę w realnym scenariuszu programowania światowego. Szczęśliwie dla nas, środowiskiem, które zamierzamy użyć, jest Processing, darmowe i otwarte oprogramowanie opracowane przez Bena Fry i Caseya Reasa w MIT Media Lab w 2001 roku. Podstawowa biblioteka funkcji do rysowania grafiki na ekranie zapewni natychmiastową wizualną informację zwrotną i wskazówki co do tego, co robi kod. A ponieważ język programowania wykorzystuje wszystkie te same zasady, struktury i koncepcje innych języków (w szczególności Java), wszystko, czego uczysz się za pomocą Processing, to prawdziwe programowanie. To nie jest jakiś udawany język, który pomoże ci zacząć; ma wszystkie podstawowe i podstawowe pojęcia, które mają wszystkie języki. Po przeczytaniu tego i nauce programowania, możesz nadal używać przetwarzania w swoim życiu akademickim lub zawodowym jako narzędzia do tworzenia prototypów lub produkcji. Możesz także zabrać zdobytą wiedzę i zastosować ją do nauki innych języków i środowisk tworzenia treści. Możesz w rzeczywistości odkryć, że programowanie nie jest twoją filiżanką herbaty; niemniej jednak poznanie podstaw pomoże ci stać się lepiej poinformowanym obywatelem technologii podczas pracy nad projektami współpracy z innymi projektantami i programistami. Może wydawać się przesadą, aby podkreślić, dlaczego w odniesieniu do przetwarzania. W końcu niniejszy tekst skupia się przede wszystkim na poznaniu podstaw programowania komputerowego w kontekście grafiki komputerowej i projektowania. Ważne jest jednak, aby zastanowić się nad przyczynami wyboru języka programowania książki, klasy, zadania domowego, aplikacji internetowej, pakietu oprogramowania i tak dalej. W końcu, teraz, kiedy zaczniesz nazywać siebie programistą na przyjęciach koktajlowych, to pytanie pojawi się w kółko. Potrzebuję programowania, aby zrealizować projekt X, z jakiego języka i środowiska powinienem korzystać? Mówię bez cienia wątpliwości, że dla ciebie, początkującego, odpowiedź brzmi: Przetwarzanie. Jego prostota jest idealna dla początkujących. Pod koniec tego rozdziału będziesz gotowy do pracy przy pierwszym projekcie obliczeniowym i gotowy do nauki podstawowych pojęć programowania. Ale prostota nie jest tam, gdzie kończy się przetwarzanie. Podróż przez wystawę Online Processing (http://processing.org/exhibition/) pozwoli odkryć szeroką gamę pięknych i innowacyjnych projektów opracowanych w całości z Processing. Pod koniec tej książki będziesz miał wszystkie narzędzia i wiedzę, których potrzebujesz, aby zrealizować swoje pomysły i przekształcić je w projekty oprogramowania realnego świata, takie jak te znalezione na wystawie. Przetwarzanie jest świetne zarówno do nauki, jak i do produkcji, istnieje bardzo niewiele innych środowisk i języków, o których można powiedzieć.

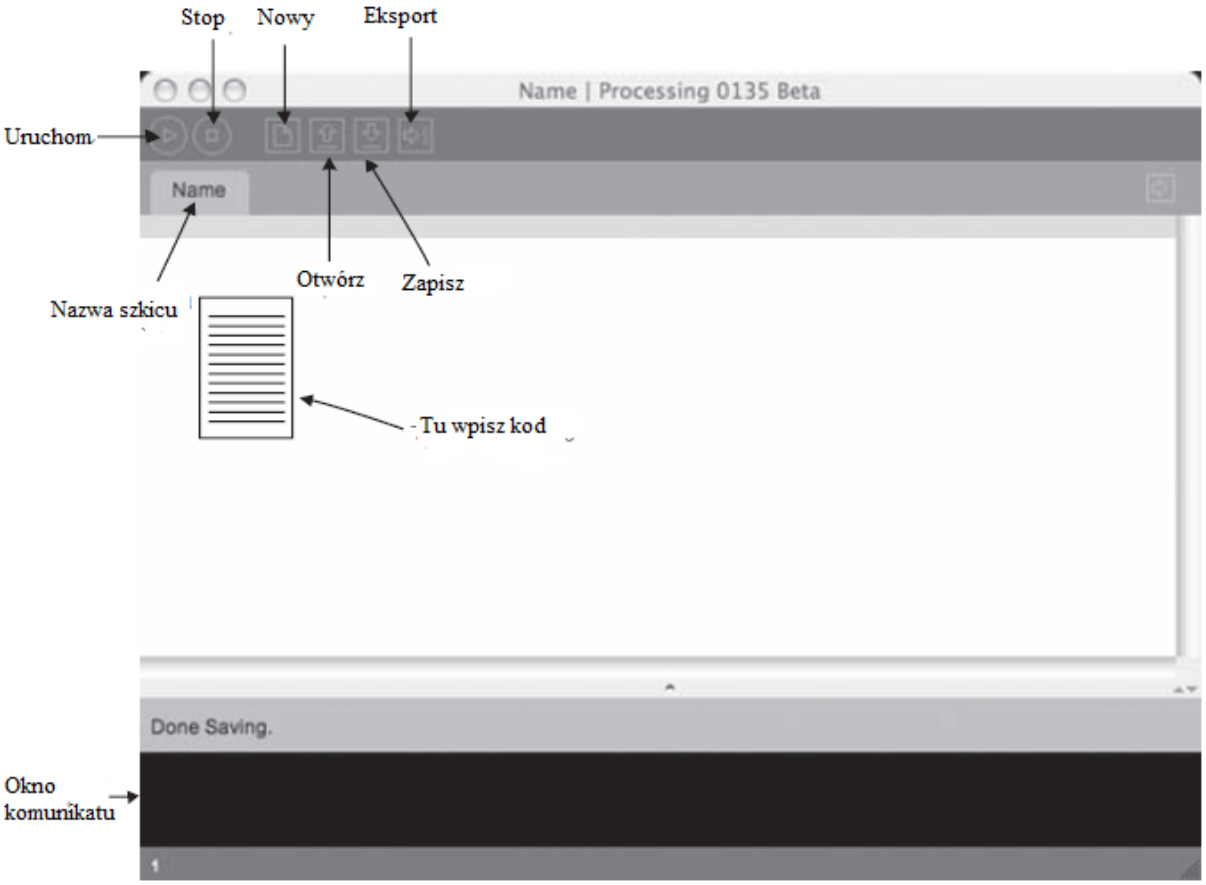
2.2 Jak uzyskać Processing?

W przeważającej części ten tekst zakłada, że masz podstawową praktyczną wiedzę na temat obsługi komputera osobistego. Dobrą wiadomością jest oczywiście to, że Processing jest dostępny do bezpłatnego pobrania. Przejdź na stronę http://www.processing.org/ i odwiedź stronę pobierania. Jeśli jesteś użytkownikiem systemu Windows, zobaczysz dwa opcje: "Windows (standard)" i "Windows (ekxert)" .Kiedy czytasz ten tekst jest całkiem prawdopodobne, że jesteś początkujący, w takim przypadku będziesz potrzebować wersji standardowej. Wersja ekspert przeznaczona jest dla tych, którzy sami już zainstalowali Javę. W systemie Mac OS X dostępna jest tylko jedna opcja pobierania. Dostępna jest również wersja Linux. Systemy operacyjne i programy zmieniają się oczywiście, więc jeśli ten akapit jest przestarzały lub przestarzały, odwiedź stronę pobierania na stronie, aby uzyskać informacje na temat tego, czego potrzebujesz. Oprogramowanie przetwarzające pojawi się jako skompresowany plik. Wybierz ładny katalog do przechowywania aplikacji (zazwyczaj "c: \ Program Files \" w systemie Windows i "Aplikacje" na komputerze Mac), wyodrębnij pliki tam, zlokalizuj plik wykonywalny "Processing" i uruchom go.

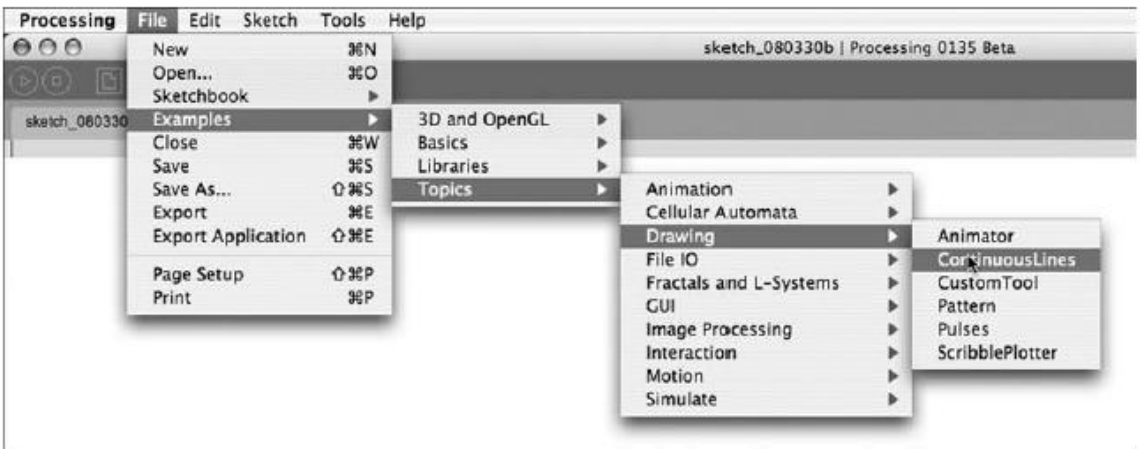
Ćwiczenie 2-1: Pobierz i zainstaluj przetwarzanie.

2.3 Aplikacja Processing

Środowisko programistyczne Processing jest uproszczonym środowiskiem do pisania kodu komputerowego i jest tak proste, jak proste oprogramowanie do edycji tekstu (takie jak TextEdit lub Notepad) połączone z odtwarzaczem multimedialnym. Każdy szkic (programy obróbki nazywane są "szkicami") ma nazwę, miejsce, w którym można wpisać kod, oraz kilka przycisków do zapisywania, otwierania i uruchamiania szkiców. Zobacz rysunek



Aby upewnić się, że wszystko działa, dobrze jest spróbować uruchomić jeden z przykładów przetwarzania. Idź do FILE → EXAMPLES → (wybierz przykład, sugerowany: Drawing → ContinuousLines), jak pokazano na rysunku poniżej



Po otwarciu przykładu kliknij przycisk "Run", jak pokazano na rysunku:

Jeśli pojawi się nowe okno z uruchomionym przykładem, wszystko gotowe! Jeśli to nie nastąpi, odwiedź FAQ online "Przetwarzanie się nie rozpocznie! "Dla możliwych rozwiązań. Stronę tę można znaleźć pod tym bezpośrednim linkiem: http://www.processing.org/faq/bugs.html # wontstart.

Ćwiczenie 2-2: Otwórz szkic z przykładów przetwarzania i uruchom go.

Programy Processing można również wyświetlać w trybie pełnoekranowym (określanym jako "tryb bieżący" w Processing). Jest to dostępne poprzez opcję menu: Sketch → Present (lub przez kliknięcie Shift przy przycisku Run). Present nie zmieni rozmiaru ekranu. Jeśli chcesz, aby szkic obejmował cały ekran, musisz użyć wymiarów ekranu w size().

2.4 Szkicownik

Programy przetwarzania są nieformalnie nazywane szkicami, w duchu szybkiego prototypowania grafiki, a my zatrudnimy ten termin w trakcie tego tektu. Folder, w którym przechowujesz szkice, nazywany jest twoim "szkicownikiem". Z technicznego punktu widzenia, po uruchomieniu szkicu w procesie przetwarzania, działa on jako aplikacja lokalna na twoim komputerze. Jak zobaczymy później, Processing pozwala również eksportować szkice jako aplety internetowe (mini-programy, które są osadzone w przeglądarce) lub jako samodzielne aplikacje specyficzne dla platformy (które mogłyby na przykład zostać udostępnione do pobrania). Po upewnieniu się, że przykłady przetwarzania działają, możesz zacząć tworzyć własne szkice. Kliknięcie przycisku "nowy" spowoduje wygenerowanie pustego nowego szkicu o nazwie według daty .Dobrym pomysłem jest zapisać jako "Zapisz jako" i utwórz własną nazwę szkicu. (Uwaga: Processing nie pozwala na spacje ani łączniki, a nazwa szkicu nie może zaczynać się od cyfry). Po pierwszym uruchomieniu Processing utworzono domyślny katalog "Processing", w którym przechowywane są wszystkie szkice w folderze "Moje dokumenty" w systemie Windows oraz w "Dokumenty" w systemie OS X. Chociaż można wybrać dowolny katalog na dysku twardym, ten folder jest domyślny. Jest to całkiem niezły folder do użycia, ale można go zmienić, otwierając preferencje Przetwarzania (dostępne w menu PLIKU). Każdy szkic Przetwarzania składa się z folderu (o tej samej nazwie co szkic) i pliku z rozszerzeniem "pde. "Jeśli twój szkic obróbki nosi nazwę MyFirstProgram, będziesz miał folder o nazwie MyFirstProgram z plikiem MyFirstProgram.pde wewnątrz. Plik "pde" to zwykły plik tekstowy zawierający kod źródłowy. (Później zobaczymy, że szkice Processing mogą mieć wiele pde.) Niektóre szkice zawierają również folder o nazwie "data", w którym są używane elementy multimedialne w programie, takie jak pliki obrazów, klipy dźwiękowe, a więc włączone, są przechowywane.

Ćwiczenie 2-3: Wpisz niektóre instrukcje z Rozdziału 1 w pusty szkic. Zwróć uwagę na kolor niektórych słów. Uruchom szkic. Czy robi to, o czym myślałeś?

2.5 Kodowanie w Processing

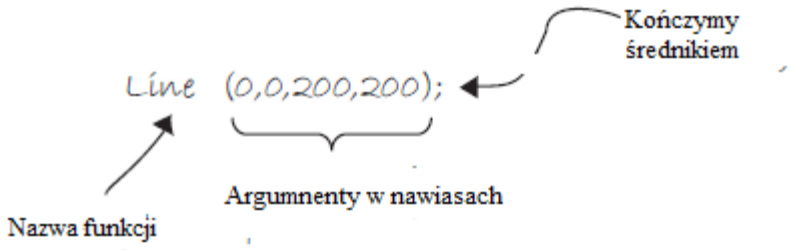
Później należy zacząć pisać jakiś kod, używając elementów omówionych w rozdziale 1. Przeanalizujmy podstawowe zasady składni. Są trzy rodzaje zdań, które możemy napisać:

• Wywołania funkcji

• Operacje przypisania

• Struktury kontrolne

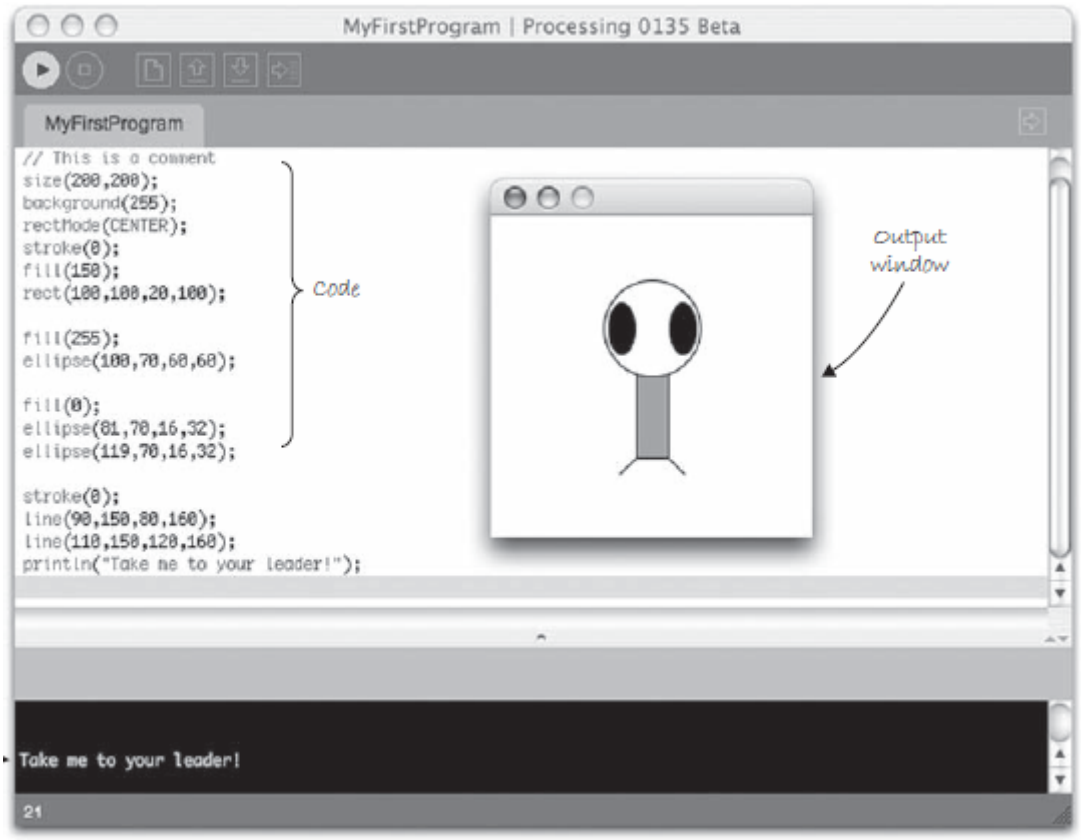
Na razie każda linia kodu będzie wywoływać funkcję. Zobacz rysunek poniżej .



W kolejnych częściach przeanalizujemy pozostałe dwie kategorie. Funkcje mają nazwę, a następnie zestaw argumentów ujęty w nawiasy. Przywołując część 1, użyliśmy funkcji, aby opisać, jak narysować kształty (nazwaliśmy je po prostu "komendami" lub "instrukcjami"). Myśląc o wywołaniu funkcji w języku naturalnym, nazwa funkcji jest czasownikiem ("draw"), a argumenty są obiektami ("punkt 0,0") zdania. Każde wywołanie funkcji musi zawsze kończyć się średnikiem. Poznaliśmy już kilka funkcji, w tym background (), stroke (), fill (), noFill (), noStroke (), point (), line (), rect (), ellipse (), rectMode () i ellipseMode () .Processing wykona sekwencję funkcji jeden po drugim i zakończy się wyświetlając narysowany wynik w oknie. Zapomnieliśmy jednak nauczyć się jednej bardzo ważnej funkcji w rozdziale 1, jednak-size (). size () określa wymiary okna, które chcesz utworzyć i bierze dwa argumenty, szerokość i wysokość. Funkcja size () powinna zawsze być pierwsza.

size (320,240); : Otwórz okno o szerokości 320 i wysokości 240 pikseli

Napiszmy pierwszy przykład



Jest kilka dodatkowych rzeczy do zapamiętania.

• Edytor tekstu Processing będzie oznaczał na kolorowo słowa (czasami nazywane słowami "zarezerwowanymi" lub "słowami kluczowymi"). Te słowa to na przykład funkcje rysowania dostępne w bibliotece Processing, zmienne "wbudowane" i stałych, a także niektórych słów, które są dziedziczone z języka programowania Java.

• Czasami przydatne jest wyświetlanie informacji tekstowych w oknie przetwarzania wiadomości (umieszczonym na dole). Dokonuje się tego za pomocą funkcji println(). println() przyjmuje jeden argument, ciąg znaków ujęty w cudzysłów. Po uruchomieniu programu Processing wyświetla ten String w oknie wiadomości iw tym przypadku String to "Zabierz mnie do swojego lidera! "Ta możliwość drukowania do okna wiadomości jest przydatna przy próbie debugowania wartości zmiennych.

• Liczba w lewym dolnym rogu wskazuje, który numer linii w kodzie jest wybrany.

• Możesz pisać "komentarze" w swoim kodzie. Komentarze są liniami tekstu, które przetwarzanie ignoruje podczas działania programu. Powinieneś użyć ich jako przypomnienia o tym, co oznacza kod, błąd, który chcesz naprawić, lub listę rzeczy do wstawienia, i tak dalej. Komentarze do pojedynczej linii są tworzone z dwoma ukośnikami, //. Komentarze do wielu linii są oznaczone znakiem / *, a następnie komentarzem i kończącym się znakiem * /.

// To jest komentarz jednoliiowe

/ * To jest komentarz, który

obejmuje kilka linii

kodu * /

Szybkie słowo na temat komentarzy. Powinieneś mieć zwyczaj pisania komentarzy w swoim kodzie. Nawet jeśli nasze szkice będą bardzo proste i krótkie, powinieneś umieścić komentarze na wszystko. Kod jest bardzo trudny do odczytania i zrozumienia bez komentarzy. Nie musisz mieć komentarza do każdego wiersza kodu, ale im więcej je dodasz, tym łatwiej będzie Ci później dokonać przeglądu i ponownego użycia kodu. Komentarze zmuszają również do zrozumienia, jak działa kod w trakcie programowania. Jeśli nie wiesz, co robisz, jak możesz napisać komentarz na ten temat? Komentarze nie zawsze będą zawarte w tekście tutaj. Dzieje się tak, ponieważ stwierdzam, że w przeciwieństwie do rzeczywistego programu, komentarze do kodu są trudne do odczytania tutaj. Zamiast tego, będziemy często używać "podpowiedzi" do kodu, aby uzyskać dodatkowe informacje i wyjaśnienia. Jeśli spojrzysz na przykłady książki na stronie internetowej, komentarze będą zawsze uwzględniane. Nie mogę tego wystarczająco podkreślić, piszę komentarze!

// Komentarz na temat tego kodu

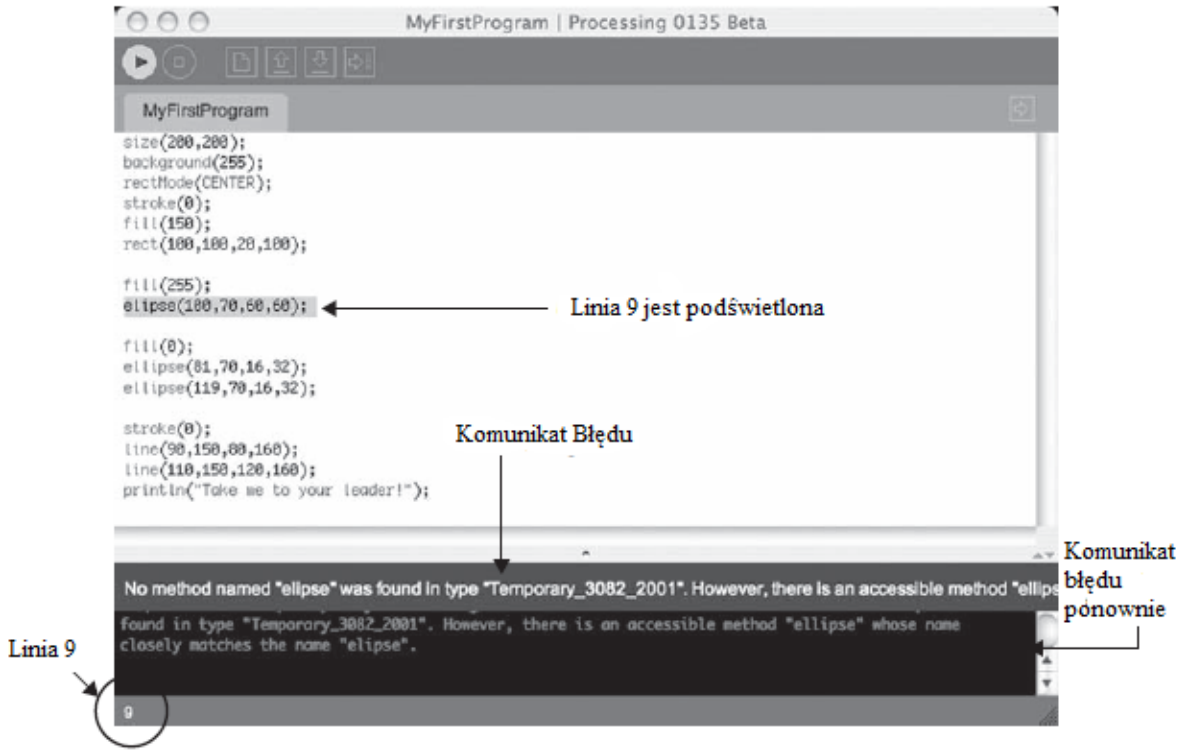
line (0,0, 100,100); : Podpowiedź do tego kodu

Ćwiczenie 2-4: Utwórz pusty szkic. Weź kod z końca rozdziału 1 i wpisz go w oknie Processing. Dodaj komentarze, aby opisać, co robi kod. Dodaj instrukcję println(), aby wyświetlić tekst w oknie komunikatu. Zapisz szkic. Naciśnij przycisk "Uruchom". Czy to działa, czy pojawia się błąd?

2.6 Błędy

Poprzedni przykład działa tylko dlatego, że nie popełniliśmy żadnych błędów ani literówek. W ciągu życia programisty jest to zjawisko dość rzadkie. W większości przypadków nasze pierwsze naciśnięcie przycisku odtwarzania nie zakończy się sukcesem. Zbadajmy, co się dzieje, gdy popełnimy błąd w

naszym kodzie na rysunku poniżej. Rysunek pokazuje, co dzieje się, gdy wpiszesz - "elipse" zamiast "elipsa" na linii 9. Jeśli wystąpi błąd w kodzie po naciśnięciu przycisku odtwarzania, przetwarzanie nie otworzy okna szkicu, a zamiast tego wyświetli Komunikat o błędzie. Jest to szczególna wiadomość, która jest dość przyjazna, informując nas, że prawdopodobnie chcieliśmy wpisać "elipsę". "Nie wszystkie komunikaty o błędach przetwarzania są tak łatwe do zrozumienia, i będziemy nadal przyglądać się innym błędom w trakcie tego tekstu



Processing uwzględnia wielkość liter!

Jeśli wpiszesz Elipsa zamiast elipsy, zostanie to również uznane za błąd.

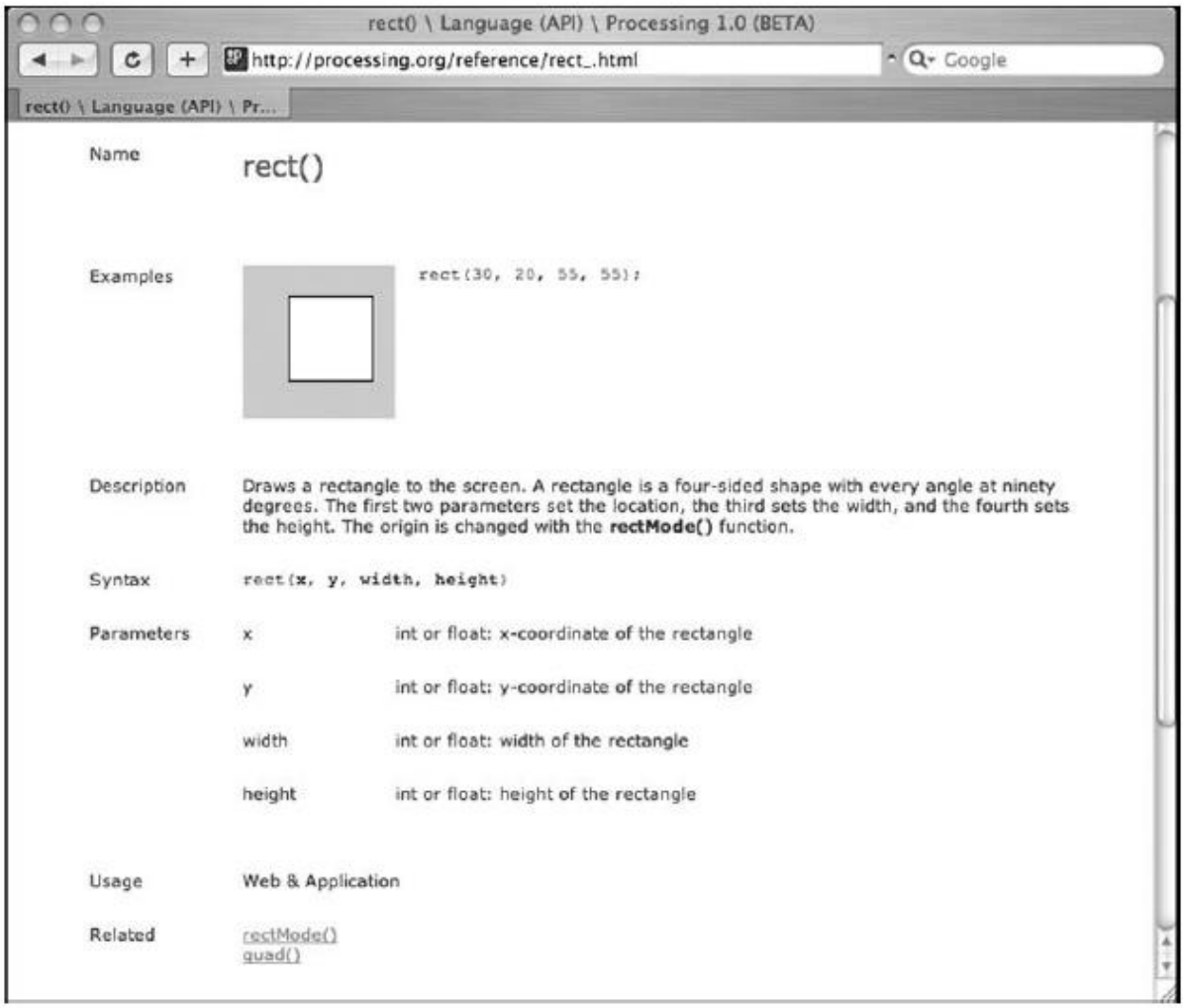
W tym przypadku wystąpił tylko jeden błąd. Jeśli pojawi się wiele błędów, Processing powiadomi Cię tylko o tym fakcie, który znajdzie (i prawdopodobnie po poprawieniu tego błędu, następny błąd zostanie wyświetlony w czasie wykonywania). Jest to trochę niefortunne ograniczenie, ponieważ często przydaje się dostęp do całej listy błędów przy ustalaniu programu. Jest to po prostu jeden z kompromisów uzyskanych w uproszczonym środowisku, takim jak Processing. Nasze życie jest prostsze, ponieważ musimy tylko spojrzeć na jeden błąd na raz, nie mamy jednak dostępu do pełnej listy. Faktem jest, że dopiero dalej podkreśla się znaczenie stopniowego rozwoju omawianego we wstępie . Jeśli tylko zaimplementujemy tylko jedną funkcję naraz, możemy popełnić tylko jeden błąd na raz.

Ćwiczenie 2-5: Staraj się celowo popełniać błędy. Czy komunikaty o błędach są zgodne z oczekiwaniami?

2.7 Odniesienie do przetwarzania

Funkcje, które zademonstrowaliśmy - ellipse(), line(), stroke() i tak dalej - są częścią biblioteki Processing. Skąd wiemy, że "elipsa" nie jest pisana "elipse", czy też ,że rect () przyjmuje cztery argumenty ("współrzędna x", "współrzędna y", "szerokość" i "wysokość")? Wiele z tych szczegółów jest intuicyjnych i to mówi o mocy Processing jako języka programowania dla początkujących. Niemniej

jednak, jedynym sposobem, aby się upewnić, jest przeczytanie internetowego odnośnika. Chociaż zajmiemy się wieloma elementami z odnośnika, nie jest to w żadnym wypadku substytut odnośnika i oba będą wymagane, aby nauczyć się przetwarzania. Odniesienie do Processing można znaleźć w Internecie na oficjalnej stronie internetowej (http://www.processing.org) pod linkiem "reference". Następnie możesz przeglądać wszystkie dostępne funkcje według kategorii lub alfabetycznie. Jeśli na przykład odwiedziłeś stronę dla rect (), znajdziesz wyjaśnienie przedstawione na rysunku poniżej



Jak widać, strona z odsyłaczami zawiera pełną dokumentację funkcji rect (), w tym:

• Nazwa - nazwa funkcji.

• Przykłady - Przykład kodu (i wizualnego wyniku, jeśli dotyczy).

• Opis - Przyjazny opis działania funkcji.

• Składnia - Szczegółowa składnia sposobu pisania funkcji.

• Parametry - Są to elementy, które wchodzą w nawiasy. Informuje Cię, jakie dane wstawiasz (liczba, znak itp.) I co oznacza ten element. (Stanie się jaśniejsze, gdy będziemy analizować więcej w kolejnych rozdziałach). Są one czasami określane jako "argumenty. "

• Returns - Czasami funkcja wysyła coś z powrotem do ciebie, gdy ją wywołujesz (np. zamiast pytać o funkcję wykonywania zadania takiego jak narysowanie okręgu, możesz poprosić o funkcję dodania dwóch liczby i zwrócenie odpowiedzi). Znowu stanie się to wyraźniejsze później.

• Sposób użycia - określone funkcje będą dostępne w przypadku apletów przetwarzania publikowanych online ("Web"), a niektóre będą dostępne tylko po uruchomieniu przetwarzania lokalnie na komputerze ("aplikacja").

• Powiązane metody - Lista funkcji często wywoływanych w związku z bieżącą funkcją. Zauważ, że "funkcje" w Javie są często nazywane "metodami".

Processing ma również bardzo przydatną opcję "znajdź w odnośniku". Kliknij dwukrotnie dowolne słowo kluczowe, aby je wybrać, i przejdź do opcji HELP → FIND IN REFERENCE (lub wybierz słowo kluczowe i naciśnij SHIFT + CNTRL + F).

Ćwiczenie 2-7: Korzystając z referencji Processing, spróbuj wdrożyć dwie funkcje, których jeszcze nie omówiliśmy w tej książce. Pozostań w kategoriach "Shape" i "Color (setting)".

Ćwiczenie 2-8: Korzystając z referencji, znajdź funkcję, która pozwala zmienić grubość linii. Jakie argumenty przyjmuje funkcja? Napisz przykładowy kod, który rysuje linię o szereokości jednego piksela , a następnie pięć pikseli szerokości, a następnie 10 pikseli szerokości.

2.8 Przycisk "Play"

Jedną z zalet jakości Processing jest to, że wystarczy uruchomić program i nacisnąć przycisk "Play". Jest to fajna metafora, a założeniem jest, że jesteśmy zadowoleni z idei grania animacji, filmów, muzyki i innych form mediów. Programy przetwarzające są nośnikami wyjściowymi w postaci grafiki komputerowej w czasie rzeczywistym, więc dlaczego po prostu ich nie odtwarzać? Niemniej jednak ważne jest, aby poświęcić chwilę i rozważyć fakt, że to, co tu robimy, nie jest tym samym, co dzieje się na iPodzie lub TiVo. Programy Processing rozpoczynają się jako tekst, są tłumaczone na kod maszynowy, a następnie uruchamiane w celu uruchomienia. Wszystkie te kroki następują kolejno po naciśnięciu przycisku odtwarzania. Przeanalizujmy te kroki jeden po drugim, rozluźniając się świadomością, że Processing zajmuje się trudniejszymi aspektami:

Krok 1. Przetłumacz na język Java. Processing to naprawdę Java. Aby twój kod działał na twoim komputerze, najpierw musisz go przetłumaczyć na kod Java.

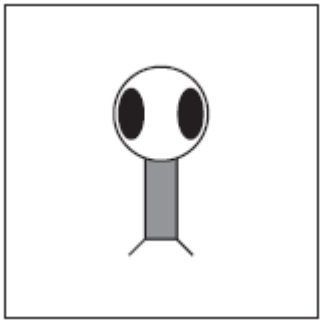
Krok 2. Skompiluj kod bajtowy Java. Kod Java utworzony w kroku 1 jest po prostu kolejnym plikiem tekstowym (z rozszerzeniem .java zamiast .pde). Aby komputer to zrozumiał, musi zostać przetłumaczone na język maszynowy. Proces tłumaczenia jest znany jako kompilacja. Jeśli programowałeś w innym języku, np. C, kod został skompilowany bezpośrednio do specyfikacji systemu operacyjnego komputera. W przypadku Javy kod jest kompilowany do specjalnego języka maszyny znanego jako kod bajtowy Java. Może działać na różnych platformach (Mac, Windows, telefony komórkowe, PDA itp.), O ile na komputerze działa "Wirtualna maszyna Java". "Chociaż ta dodatkowa warstwa może czasami powodować, że programy działają wolniej niż mogłyby być inaczej, to wieloplatformowość jest świetną cechą Javy. Aby dowiedzieć się więcej o tym, jak to działa, odwiedź http://java.sun.com

Krok 3. Wykonanie. Skompilowany program kończy się w pliku JAR. JAR to plik archiwum Java zawierający skompilowane programy Java ("klasy"), obrazy, czcionki i inne pliki danych. Plik JAR jest wykonywany przez wirtualną maszynę Java i powoduje pojawienie się okna wyświetlacza.

2.9 Twój pierwszy szkic

Po pobraniu i zainstalowaniu Processing rozumiemy podstawowe menu i elementy interfejsu oraz zapoznaliśmy się z referencją online, jesteśmy gotowi do rozpoczęcia kodowania. Jak wspominałem krótko w części 1, pierwsza połowa tej książki będzie następować po jednym przykładzie, który ilustruje

podstawowe elementy programowania: zmienne, tablice, warianty, pętle, funkcje i obiekty. Inne przykłady zostaną dołączone po drodze, ale po jednym pokażą, w jaki sposób podstawowe elementy stojące za programowaniem komputerowym opierają się na sobie nawzajem. Przykład ten będzie oparty na historii naszego nowego przyjaciela stworka, zaczynając od statycznego renderowania o prostych kształtach. Rozwój Stworka obejmie interakcję z myszą, ruch i klonowanie do populacji wielu Stworków. Chociaż nie jesteś w żaden sposób zobowiązany do ukończenia każdego ćwiczenia tego tekst w swojej własnej, obcej formie, sugeruję, abyś zaczął od projektowania i po każdym rozdziale, rozszerzyć funkcjonalność tego projektu o koncepcje programistyczne, które są badane. Jeśli brakuje ci pomysłu, po prostu narysuj własnego małego kosmitę, nazwij go Gooz i zacznij programować! Patrz rysunek



Ćwiczenie 2-1

size(200,200); // Ustaw rozmiar okna

background(255); // Narysuj czarne tło

smooth(); : Funkcja smooth() umożliwia "wygładzanie", które wygładza krawędzie kształtu; no smooth() wyłącza wygładzanie

// Ustaw elipse i rects na tryb CENTER

ellipseMode(CENTER);

rectMode(CENTER);

// Rysuj ciało Stworka

stroke(0);

fill(150);

rect(100,100,20,100);

// Narysuj głowę Stworka

fill(255);

ellipse(100,70,60,60);

// Narysuj oczy Stworka

fill(0);

ellipse(81,70,16,32);

ellipse(119,70,16,32);

```
// Narysuj nogi Stworka

stroke(0);

line(90,150,80,160);

line(110,150,120,160);
```

Udawajmy, przez chwilę, że ten projekt Zooga jest tak zadziwiająco wspaniały, że nie można się doczekać, aż pojawi się on na ekranie komputera. (Tak, zdaję sobie sprawę, że może to wymagać dość znaczącego zawieszenia niewiary). Aby uruchomić dowolny i wszystkie przykłady kodu znalezione w tej książce, masz dwie możliwości:

• Wpisz ponownie kod ręcznie.

• Odwiedź stronę internetową książki (http://www.learningprocessing.com), znajdź przykład po numerze i skopiuj / wklej (lub pobierz) kod.

Z pewnością opcja nr 2 jest łatwiejsza i mniej czasochłonna, a ja polecam korzystanie z witryny jako zasobu do oglądania szkiców działających w czasie rzeczywistym i chwytania przykładów kodu. Niemniej jednak, gdy zaczniesz się uczyć, istnieje prawdziwa wartość w samodzielnym wpisywaniu kodu. Twój mózg będzie podbijał składnię i logikę podczas pisania, a nauczysz się wiele, popełniając błędy po drodze. Nie wspominając o tym, że po wprowadzeniu każdego nowego wiersza kodu po prostu uruchamia szkic, eliminując wszelkie tajemnice dotyczące działania szkicu. Najlepiej będziesz wiedzieć, kiedy będziesz gotowy do kopiowania / wklejania. Śledź swoje postępy i jeśli zaczniesz wyświetlać wiele przykładów bez poczucia komfortu z ich działaniem, spróbuj wrócić do instrukcji pisania na maszynie.

Ćwiczenie 2-9: Korzystając z tego, co zaprojektowałeś w rozdziale 1, zaimplementuj własny rysunek ekranowy, używając tylko podstawowych kształtów 2D - arc(), curve(), ellipse(), line (), point (), quad (), rect (), triangle () - i podstawowe funkcje kolorów - background(), colorMode(), fill(), noFill (), noStroke () i stroke (). Pamiętaj, aby użyć size () do określenia wymiarów okna. Wskazówka: Zagraj w szkic po wpisaniu każdego nowego wiersza kodu. Popraw wszelkie błędy lub literówki po drodze.

2.10 Publikowanie programu

Po ukończeniu szkicu Processing, można go opublikować w Internecie jako aplet Java. To będzie bardziej ekscytujące, gdy będziemy tworzyć interaktywne, animowane aplety, ale dobrze jest praktykować z prostym przykładem. Po zakończeniu ćwiczenia 2-9 i stwierdzeniu, że szkic działa, wybierz FILE → EXPORT. Zwróć uwagę, że jeśli masz błędy w swoim programie, nie eksportuje się prawidłowo, więc zawsze testuj, uruchamiając najpierw! Nowy katalog o nazwie "applet" zostanie utworzony w folderze szkicu i wyświetlony, jak pokazano na rysunku

Masz teraz niezbędne pliki do opublikowania swojego apletu w Internecie.

• index.html - Źródło HTML strony wyświetlającej aplet.

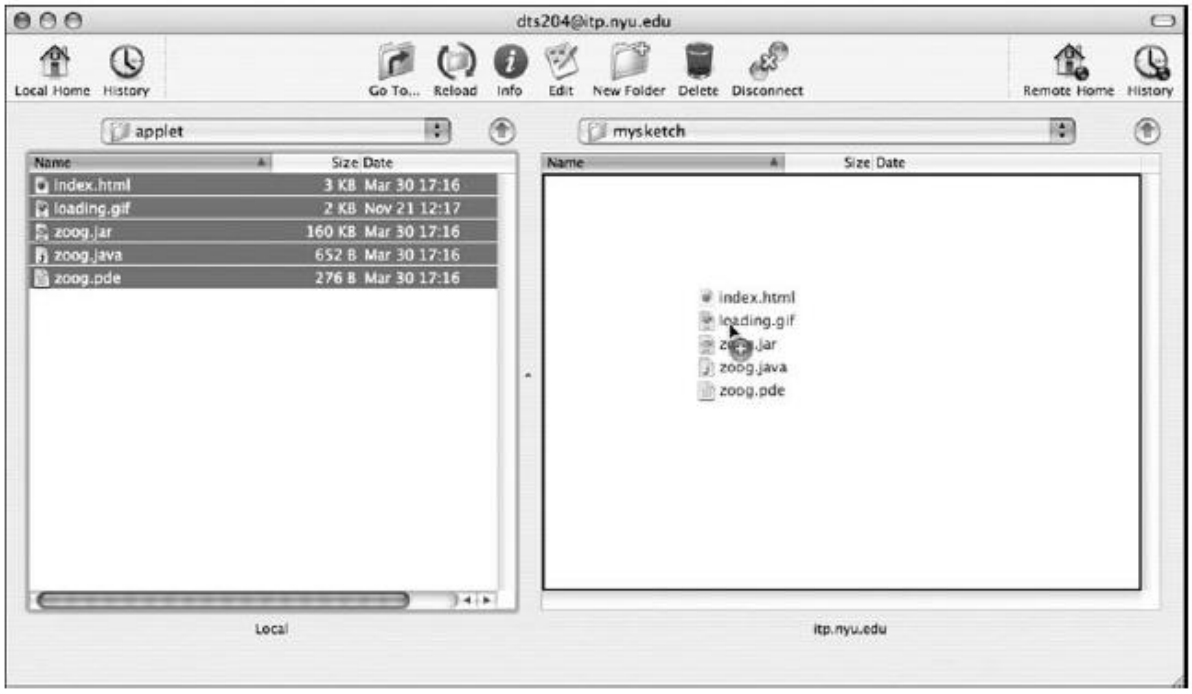
• loading.gif - Obraz wyświetlany, gdy użytkownik ładuje aplet (Przetwarzanie dostarczy domyślny, ale możesz stworzyć swój własny).

• zoog.jar - sam skompilowany aplet.

• zoog.java - przetłumaczony kod źródłowy Java (wygląda jak kod przetwarzania, ale ma kilka dodatkowych rzeczy, których wymaga Java.)

• zoog.pde - Twoje źródło Processing.

Aby zobaczyć działanie apletu, kliknij dwukrotnie plik "index.html", który powinien uruchomić stronę w domyślnej przeglądarce. Zobacz rysunek 2.10. Aby uruchomić aplet w trybie online, będziesz potrzebować miejsca na serwerze sieciowym i oprogramowania FTP (możesz też użyć strony do udostępniania szkicu przetwarzania, takiej jak http://www.openprocessing.org).



Ćwiczenie 2-10: Eksportuj szkic jako aplet. Zobacz szkic w przeglądarce (lokalnie lub zdalnie).

III. Interakcje

"Zawsze pamiętaj, że to wszystko zaczęło się od marzenia i myszy. "-Walt Disney

"Jakość wyobraźni polega na płynięciu, a nie na zamrażaniu. " -Ralph Waldo Emerson

3.1 Idź z prądem.

Jeśli kiedykolwiek grałeś w grę komputerową, wchodziłeś w interakcję z cyfrową instalacją artystyczną lub oglądałeś wygaszacz ekranu o trzeciej nad ranem, prawdopodobnie niewiele myślałeś o tym, że oprogramowanie, które uruchamia te doświadczenia, dzieje się przez pewien czas. Rozpoczyna się gra, uratujesz księżniczkę tak i tak od złego lorda, który - z-ma-co-to - osiąga wysoki wynik, a gra się kończy. W tej części chciałbym skupić się na tym bardzo "przepływie" w czasie. Gra rozpoczyna się zbiorem początkowych warunku: nazywasz swoją postać, zaczynasz od zera i zaczynasz od poziomu pierwszego. Pomyślmy o tej części jako SETUP programu. Po zainicjowaniu tych warunków rozpoczynasz grę. W każdej chwili komputer sprawdza, co robisz za pomocą myszy, oblicza wszystkie odpowiednie zachowania postaci w grze i aktualizuje ekran, aby renderować całą grafikę gry. Cykl obliczania i rysowania odbywa się w kółko, najlepiej 30 lub więcej razy na sekundę, aby uzyskać płynną animację. Pomyślmy o tej części jako DRAW programu.

Ta koncepcja ma kluczowe znaczenie dla naszej zdolności do wyjścia poza statyczne projekty z przetwarzaniem.

Krok 1. Ustaw warunki początkowe dla programu jeden raz.

Krok 2. Zrób coś w kółko i raz po raz (i ponad ...), aż program się skończy.

Zastanów się, jak możesz zacząć biegać.

Krok 1. Załóż tenisówki i rozciągnij. Zrób to raz, dobrze?

Krok 2. Połóż prawą stopę do przodu, a następnie lewą stopę. Powtarzaj to tak szybko, jak to tylko możliwe.

Krok 3. Po 26 km skończ.

Ćwiczenie 3-1: Po angielsku wypisz "przepływ" prostej gry komputerowej, takiej jak Pong. Jeśli nie znasz Ponga, odwiedź: http://en.wikipedia.org/wiki/Pong.

3.2 Nasi dobrzy przyjaciele, setup() i draw()

Teraz, gdy jesteśmy dobrzy i wyczerpani bieganiem maratonami, aby lepiej nauczyć się programowania, możemy wykorzystać tę nowo odkrytą wiedzę i zastosować ją do naszego pierwszego "dynamicznego" szkicu Processing. W przeciwieństwie do statycznych przykładów z części 2, ten program będzie rysował na ekranie w sposób ciągły (tj. aż użytkownika skończy). Dokonuje się tego poprzez napisanie dwóch "bloków kodu" setup() i draw(). Technicznie rzecz biorąc setup() i draw() są funkcjami. Wkroczymy w dłuższą dyskusję na temat pisania nasze własne funkcje w późniejszej części; na razie rozumiemy, że są to dwie sekcje, w których piszemy kod.

Co to jest blok kodu?

Blok kodu to dowolny kod zamknięty w nawiasach klamrowych.

{

Blok kodu

}

Bloki kodu mogą być również zagnieżdżone w sobie.

{

Blok kodu

{

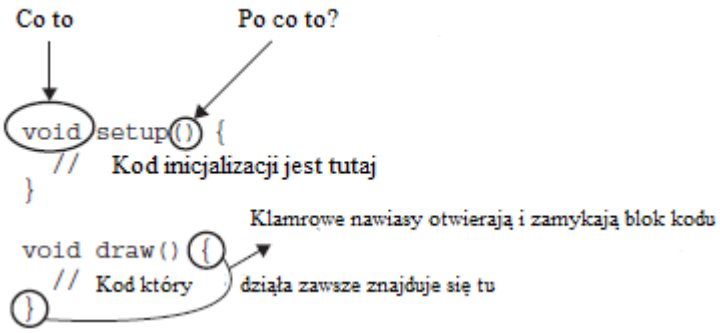
Blok wewnątrz bloku kodu

}

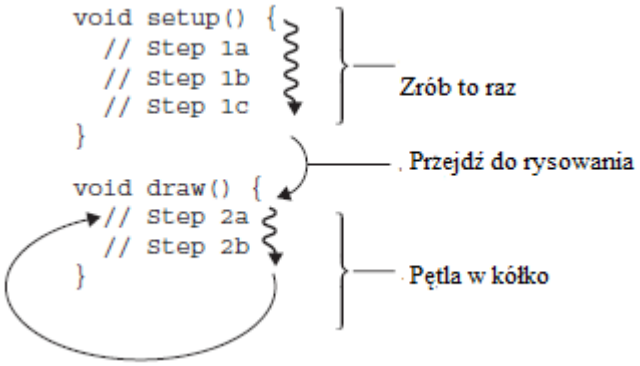
}

Jest to ważna konstrukcja, ponieważ pozwala nam oddzielić i zarządzać naszym kodem jako indywidualnymi kawałkami większej układanki. Konwencja programowania polega na wcięciu linii kodu w każdym bloku, aby kod był bardziej czytelny. Processing zrobi to za pomocą opcji Auto-Format (Tools → Auto Format). Bloki kodu ujawnią się jako kluczowe w opracowywaniu bardziej złożonej logiki, pod względem zmiennych, warunków, iteracji, obiektów i funkcji, jak omówiono w kolejnych częściach. Na razie wystarczy spojrzeć na dwa proste bloki: setup() i draw().

Spójrzmy na to, co z pewnością będzie dziwnie wyglądającą składnią dla setup() i draw().



Trzeba przyznać, że na powyższym rysunku jest dużo rzeczy, których nie jesteśmy w stanie się całkowicie nauczyć. Mówiliśmy, że nawiasy klamrowe wskazują początek i koniec "bloku kodu", ale dlaczego są nawiasy po "setup" i "draw"? Och, i moja dobroć, o co chodzi w tej "pustce"? Te ważne elementy składni zaczną mieć sens w przyszłych częściach, gdy ujawni się więcej pojęć. Na razie najważniejsze jest skupienie się na tym, w jaki sposób powyższe struktury kontrolują przepływ naszego programu. Zostało to pokazane na rysunku poniżej



Jak to działa? Gdy uruchomimy program, będzie on postępował dokładnie według naszych instrukcji, najpierw wykonując czynności w setup (), a następnie przechodząc do kroków w draw (). Zlecenie kończy się czymś w rodzaju:

1a, 1b, 1c, 2a, 2b, 2a, 2b, 2a, 2b, 2a, 2b, 2a, 2b, 2a, 2b ...

Teraz możemy przepisać przykład Stworka jako szkic dynamiczny. Patrz przykład 3

```
void setup() {
// Ustaw rozmiar okna  : setup() uruchamia się po raz pierwszy. size() powinien zawsze być pierwszym
                         wierszem konfiguracji (), ponieważ Processing nie będzie w stanie nic zrobić
                         przed wielkością okna, jeśli został określony.
size(200,200);
}
void draw() {            : draw() tworzy pętle, dopóki nie zamkniesz okna szkicu.
// Narysuj białe tło
background(255);
// Ustaw tryb CENTER
ellipseMode(CENTER);
rectMode(CENTER);
// Narysuj ciało Stworka
stroke(0);
fill(150);
rect(100,100,20,100);
// Narysuj głowę Stwork
stroke(0);
fill(255);
ellipse(100,70,60,60);
// Narysuj oczy Stworka
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
// Narysuj nogi Stowrka
stroke(0);
line(90,150,80,160);
```

```
line(110,150,120,160);

}
```

Pobierz kod z przykładu 3 i uruchom go w Processing. Dziwne, prawda? Zauważysz, że nic w oknie się nie zmienia. Wygląda identycznie jak szkic statyczny! Co się dzieje? Cała ta dyskusja za nic? Cóż, jeśli przeanalizujemy kod, zauważymy, że nic w funkcji draw() nie jest różne. Za każdym razem przez pętlę program przechodzi przez kod i wykonuje identyczne instrukcje. Tak, program działa z czasem, przerysowując okno, ale wygląda na statyczny, ponieważ za każdym razem rysuje to samo!

Ćwiczenie 3-2: Ponów rysunek utworzony na końcu rozdziału 2 jako program dynamiczny. Nawet jeśli będzie wyglądać tak samo, poczuj się lepiej w swoim osiągnięciu!

3.3 Odmiana za pomocą myszy

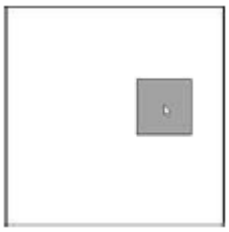
Rozważ to. Co się stanie, jeśli zamiast wpisywać liczbę w jedną z funkcji rysowania, można wpisać "lokalizację X myszy" lub "lokalizację Y myszy".

```
line (lokalizacja X myszy, lokalizacja Y myszy, 100, 100);
```

W rzeczywistości można zamiast bardziej opisowego języka, użyć słów kluczowych mouseX i mouseY, wskazując poziomą lub pionową pozycję kursora myszy.

Przykład 3-2: mouseX i mouseY

```
void setup() {

size(200,200);          :

}                                   Spróbuj przenieść background() do setup() i zobacz różnicę!

void draw() {

background(255);     :

// Body

stroke(0);

fill(175);

rectMode(CENTER);

rect(mouseX,mouseY,50,50);  :       mouseX jest słowem kluczowym, które szkic zastępuje
                                    poziomą pozycją myszy. mouseY jest słowem kluczowym,
                                    które szkic zastępuje pionową pozycją myszy.

}
```



Ćwiczenie 3-3: Wyjaśnij, dlaczego widzimy ślad prostokąta, jeśli przeniesiemy background() do setup (), pozostawiając go poza draw().

Niewidzialna linia kodu

Jeśli podążacie ściśle za logiką setup() i draw(), możecie dojść do interesującego pytania: Kiedy Procesing faktycznie wyświetla kształty w oknie? Kiedy pojawiają się nowe piksele?

Na pierwszy rzut oka można założyć, że wyświetlacz jest aktualizowany dla każdego wiersza kodu, który zawiera funkcję rysowania. Gdyby tak było, widzimy kształty pojawiające się na ekranie po jednym na raz. Będzie tak szybko, że ledwie zauważymy każdy kształt pojawiający się osobno. Jednakże, gdy okno jest wymazywane za każdym razem, gdy wywoływane jest background(), wystąpiłby pewien niefortunny i nieprzyjemny rezultat: flicker. Processing rozwiązuje ten problem, aktualizując okno tylko pod koniec każdego cyklu poprzez funkcję draw (). To tak, jakby była niewidoczna linia kodu, która renderuje okno na końcu funkcji draw ().

```
void draw () {

// Cały twój kod

// Aktualizuj okno wyświetlania - niewidoczna linia kodu, której nie widzimy

}
```

Proces ten jest znany jako podwójny bufor, a w środowisku niższego poziomu może się okazać, że musisz go zaimplementować samodzielnie. Ponownie, poświęcamy czas, aby podziękować Processing za uczynienie naszego wprowadzenia w programowanie bardziej przyjaznym i prostszym, zajmując się tym dla nas.

Moglibyśmy popchnąć ten pomysł nieco dalej i stworzyć przykład, w którym bardziej złożony wzór (wiele kształtów i kolorów) jest kontrolowany przez położenie mouseX i mouseY. Na przykład możemy przepisać Stworka, aby podążać za myszą. Zwróć uwagę, że ciało Stworka znajduje się w dokładnej lokalizacji myszy (mouseX, mouseY), jednak inne części ciała Stworka są rysowane względem myszy. Głowa Stworka, na przykład, znajduje się w (mouseX, mouseY-30). Poniższy przykład porusza jedynie ciało i głowę Stworka, jak pokazano na rysunku

```
void setup() {

size(200,200); // Ustaw rozmiar okna

smooth();

}

void draw() {

background(255); // Narysuj białe tło
```

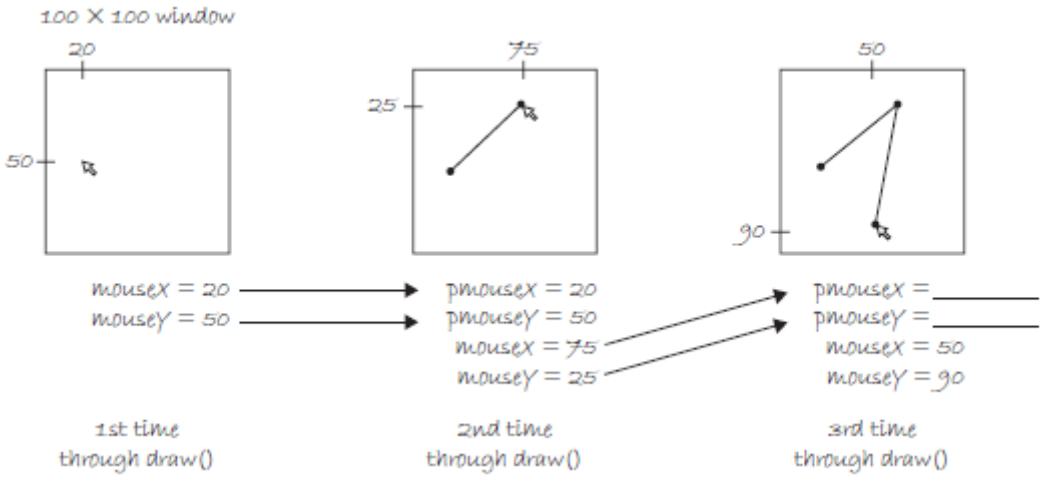
```
// Ustaw elipsy i rects na tryb CENTER
ellipseMode(CENTER);
rectMode(CENTER);
// Draw Zoog's body
stroke(0);
fill(175);
rect(mouseX,mouseY,20,100);  : Ciało Stworka jest rysowane w miejscu (mouseX, mouseY).
// Rysuj ciało Stworka
stroke(0);
fill(255);
ellipse(mouseX,mouseY-30,60,60); : Głowa Stworka jest wyciągnięta nad ciałem w miejscu (mouseX,
                                   mouseY-30)
// Rysuj oczy Stworka
fill(0);
ellipse(81,70,16,32);
ellipse(119,70,16,32);
// Rysuj nogi Stworka
stroke(0);
line(90,150,80,160);
line(110,150,120,160);
}
```

Ćwiczenie 3-4: Uzupełnij Stworka tak, aby reszta jego ciała poruszała się za pomocą myszy.

```
// Narysuj oczy Stworka
fill(0);
ellipse(_______,_______ ,16,32);
ellipse(_______,_______ ,16,32);
// Narsyuj nogi Stworka
stroke(0);
line(_______,_______,_______,_______);
line(_______,_______,_______,_______);
```

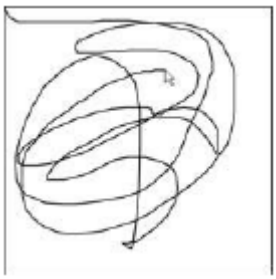
Ćwiczenie 3-5: Przekoduj swój projekt, aby kształty reagowały na mysz (zmieniając kolor i lokalizację).

Oprócz mouseX i mouseY możesz także użyć pmouseX i pmouseY. Te dwa słowa kluczowe oznaczają "poprzednie" lokalizacje myszy X i myszy Y, tj. Miejsce, w którym mysz była ostatnim razem, gdy przechodziliśmy przez funkcję draw (). To pozwala na interesujące możliwości interakcji. Na przykład rozważmy, co się stanie, jeśli narysujemy linię z poprzedniej lokalizacji myszy do aktualnej lokalizacji myszy, jak pokazano na diagramie na rysunku



Ćwiczenie 3-6: Wypełnij puste pole na rysunku 3.6.

Przez podłączenie poprzedniej lokalizacji myszy do bieżącego położenia myszy za pomocą linii za każdym razem draw(), jesteśmy w stanie wyrenderować ciągłą linię, która podąża za myszą. Zobacz rysunek



Przykład 3-4: Rysowanie linii ciągłej

```
void setup() {

size(200,200);

background(255);

smooth();

}

void draw() {

stroke(0);

line(pmouse X ,pmouse Y ,mouse X ,mouse Y ); : Narysuj linię od poprzedniej lokalizacji myszy do
                                                aktualnej lokalizacji myszy

}
```

Ćwiczenie 3-7: Formuła do obliczania prędkości ruchu poziomego myszy jest bezwzględną wartością różnicy między mouseX i pmouseX. Wartość bezwzględna liczby jest określana jako liczba bez znaku:

• Wartość bezwzględna -2 wynosi 2.

• Wartość bezwzględna 2 wynosi 2.

W Processing możemy uzyskać bezwzględną wartość liczby umieszczając ją wewnątrz funkcji abs (),

to jest,

• abs (- 5) → 5

Szybkość, z jaką porusza się mysz, wynosi zatem:

• abs (mouseX - pmouseX)

Zaktualizuj ćwiczenie 3-7, aby im szybciej poruszał się mysz, tym szersza była narysowana linia. Podpowiedź: wyszukaj wartość strokeWeight() w odnośniku Processing.

stroke(255);

_____ (_____);

line(pmouse X ,pmouse Y ,mouse X ,mouse Y);

3.4 Kliknięcia myszą i naciśnięcia klawiszy

Jesteśmy na dobrej drodze do tworzenia dynamicznych, interaktywnych szkiców przetwarzania za pomocą frameworków setup() i draw() oraz słów kluczowych mouseX i mouseY. Jednak brakuje kluczowej formy interakcji - kliknięcie myszką!

Aby dowiedzieć się, jak coś się dzieje po kliknięciu myszą, musimy wrócić do przepływu naszego programu. Wiemy, że setup () dzieje się raz, a draw () na zawsze. Kiedy pojawia się kliknięcie myszą? Myszki (i naciśnięcia klawiszy) są traktowane jako zdarzenia w przetwarzaniu. Jeśli chcemy, aby coś się stało (np. "Kolor tła zmienia się na czerwony") po kliknięciu myszą, musimy dodać trzeci blok kodu, aby obsłużyć to zdarzenie. Ta "funkcja" zdarzenia pokaże programowi, jaki kod wykonać, gdy wystąpi zdarzenie. Podobnie jak w przypadku setup (), kod pojawi się raz i tylko raz. Oznacza to, że raz i tylko raz dla każdego wystąpienia zdarzenia. Na wydarzenie, takie jak kliknięcie myszą, może się zdarzyć wiele razy! Są to dwie nowe funkcje, których potrzebujemy:

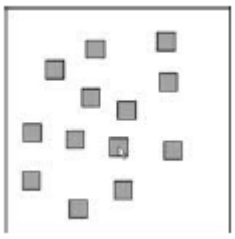
• mousePressed () - Obsługuje kliknięcia myszą.

• keyPressed () -Zawiera naciśnięcia klawiszy.

W poniższym przykładzie zastosowano obie funkcje zdarzeń, dodając kwadraty po naciśnięciu myszy i czyszczenie tła po naciśnięciu klawisza.

Przykład 3-5: mousePressed() i keyPressed()

```
void setup() {

size(200,200);

background(255);

}

void draw() {              : W tym przykładzie nic się nie dzieje w draw()!

}

void mousePressed() {  : Za każdym razem, gdy użytkownik kliknie myszką, wykonywany jest kod
                           napisany wewnątrz mousePressed (

stroke(0);

fill(175);

rectMode(CENTER);

rect(mouseX,mouseY,16,16);

}

void keyPressed() {       : Za każdym razem, gdy użytkownik naciśnie klawisz, wykonywany jest kod
                           napisany wewnątrz funkcji keyPressed ()

background(255);

}
```



W przykładzie 3-5 mamy cztery funkcje opisujące przepływ programu. Program rozpoczyna się w setup (), gdzie rozmiar i tło są inicjowane. Ciągnie się do draw(), bez końca w pętli. Ponieważ funkcja draw() nie zawiera kodu, okno pozostanie puste. Jednak dodaliśmy dwie nowe funkcje: mousePressed() i keyPressed(). Kod wewnątrz tych funkcji siedzi i czeka. Kiedy użytkownik kliknie myszą (lub naciśnie klawisz), uruchamia się, wykonując załączony blok instrukcji raz i tylko jeden raz.

Ćwiczenie 3-8: Dodaj "background(255); " do funkcji draw(). Dlaczego program przestaje działać?

Jesteśmy teraz gotowi połączyć wszystkie te elementy razem dla Stworka.

• Całe ciało Stworka podąży za myszą.

• Kolor oczu Stworka zostanie określony przez lokalizację myszy.

• Nogi Stworka będą rysowane od poprzedniej lokalizacji myszy do aktualnej lokalizacji myszy.

• Po kliknięciu myszką w oknie wiadomości pojawi się komunikat: "Zabierz mnie do swojego lidera! "

Zauważ dodanie w Przykładzie 3-6 funkcji frameRate(). frameRate(), która wymaga liczby całkowitej od 1 do 60, wymusza szybkość, z jaką przetwarzanie będzie cyklicznie przez draw(). frameRate(30), na przykład, oznacza 30 klatek na sekundę, tradycyjną prędkość animacji komputerowej. Jeśli nie uwzględnisz frameRate(), Processing spróbuje uruchomić szkic z prędkością 60 klatek na sekundę. Ponieważ komputery działają z różnymi prędkościami, frameRate() jest używany, aby upewnić się, że szkic jest spójny na wielu komputerach. Ta liczba klatek jest jednak maksymalna. Jeśli szkic ma narysować milion prostokątów, ukończenie cyklu rysowania może trwać dość długo i działać z mniejszą prędkością.

Przykład 3-6: Interaktywny Stworek

```
void setup() {

// Set the size of the window

size(200,200);

smooth();

frameRate(30) ;  : Liczba klatek na sekundę jest ustawiona na 30 klatek na sekundę.

}

void draw() {

// Draw a black background

background(255);

// Set ellipses and rects to CENTER mode

ellipseMode(CENTER);

rectMode(CENTER);

// Draw Zoog's body

stroke(0);

fill(175);

rect(mouseX,mouseY,20,100);

// Draw Zoog's head

stroke(0);

fill(255);

ellipse(mouse X ,mouse Y -30,60,60);

// Draw Zoog's eyes                  :  Kolor oczu jest określony przez lokalizację myszy.

fill(mo useX,0,mouseY);

ellipse(mouse X-19,mouseY-30,16,32);
```

```
ellipse(mouse X + 19,mouse Y-30,16,32);

// Draw Zoog's legs

stroke(0);

line(mouse X-10,mouseY + 50,pmouse X-10,pmouseY + 60); : Nogi są rysowane zgodnie z lokalizacją
                                                          myszy i poprzednią lokalizacją myszy.

line(mouse X + 10,mouse Y + 50,pmouse X + 10,pmouse Y + 60);

}

void mousePressed() {

println( "Take me to your leader! ");

}
```

Projekt Numer Jeden

(Możesz już ukończyć wiele z tego projektu poprzez ćwiczenia w częściac 1-3. Ten projekt łączy wszystkie elementy razem. Możesz zacząć od zera nowy projekt lub wykorzystanie elementów z ćwiczeń.)

Krok 1. Zaprojektuj rysunek statyczny za pomocą kolorów RGB i kształtów podstawowych.

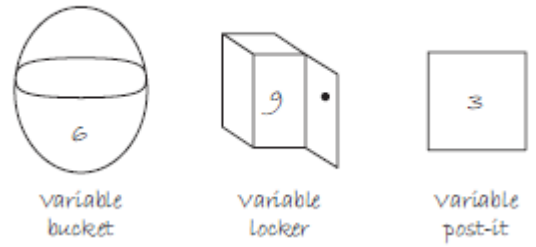
Krok 2. Wykonaj dynamiczne rysowanie ekranu statycznego poprzez interakcję z myszą. Może zawierać kształty następujące po myszy, zmieniając ich rozmiar zgodnie z myszą, zmieniając ich kolor zgodnie z myszą i tak dalej.

Użyj przestrzeni podanej poniżej, aby szkicować projekty, notatki i pseudokod dla projektu.

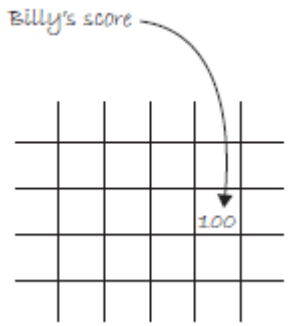
IV. Zmienne

4.1 Co to jest zmienna?

Przyznaję. Kiedy uczę programowania, przystępuję do opisu analogii, próbując w intuicyjny sposób wyjaśnić pojęcie zmiennej. Każdego dnia mogę powiedzieć: "Zmienna jest jak wiadro. "Włożyłeś coś do wiadra, nosisz ze sobą i odzyskujesz , gdy tylko poczujesz się zainspirowany. "Zmienna jest jak szafka do przechowywania. "Złóż trochę informacji w szafce, w której może bezpiecznie żyć, łatwo dostępna w każdej chwili. "Zmienna jest uroczą, żółtą karteczką, na której jest napisane: jestem zmienną. Napisz na mnie swoje informacje. "

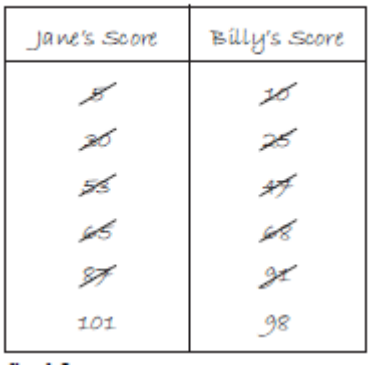


Mógłbym iść dalej. Ale nie będę. Myślę, że wpadłeś na ten pomysł. I nie jestem do końca pewien, czy naprawdę potrzebujemy analogii, ponieważ sama koncepcja jest dość prosta. Oto oferta. Komputer ma pamięć. Dlaczego nazywamy to pamięcią? Ponieważ jest to to, czego używa komputer do zapamiętywania potrzebnych rzeczy. Z technicznego punktu widzenia zmienna jest nazwanym wskaźnikiem do lokalizacji w pamięci komputera ("adres pamięci"), w której przechowywane są dane. Ponieważ komputery przetwarzają informacje tylko jedną instrukcję na raz, zmienna umożliwia programiście zapisywanie informacji z jednego punktu w programie i odsyła do niego w późniejszym czasie. Dla programisty przetwarzania jest to niezwykle przydatne; zmienne mogą śledzić informacje dotyczące kształtów: kolor, rozmiar, położenie. Zmienne są dokładnie tym, czego potrzebujesz, aby zmienić trójkąt z niebieskiego na fioletowy, kółko na ekranie i prostokąt skurczył się do zapomnienia. Spośród wszystkich dostępnych analogii preferuję podejście papierowe: papier milimetrowy. Wyobraź sobie, że pamięć komputera jest arkuszem papieru milimetrowego, a każda komórka na papierze milimetrowym ma adres. Dzięki pikselom nauczyliśmy się odnosić do tych komórek według numerów kolumn i wierszy. Czy nie byłoby miło, gdybyśmy mogli wymienić te komórki? Ze zmiennymi możemy. Nazwijmy jeden "Billy's Score" (zobaczymy, dlaczego nazywamy to w następnej sekcji) i nadajemy mu wartość 100. Na drodze, kiedy chcemy użyć wyniku Billy'ego w programie, nie musimy pamiętać wartość 100. Jest tam w pamięci i możemy poprosić o nią po imieniu.



Siła zmiennej nie spoczywa po prostu na zdolności zapamiętywania wartości. Cały punkt zmiennej polega na tym, że wartości te są różne i pojawiają się bardziej interesujące sytuacje, ponieważ okresowo zmieniamy tę wartość. Rozważmy grę Scrabble między Billy i Jane. Aby śledzić wynik, Jane

wyjmuje papier i ołówek, a następnie zapisuje dwie nazwy kolumn: "Billy's Score" i "Jane's Score". "Jako dwie gry, prowadzona jest liczba punktów każdego gracza pod nagłówkami. Jeśli wyobrażamy sobie, że ta gra jest wirtualnym Scrabble zaprogramowanym na komputerze, nagle możemy zobaczyć koncepcję zmiennej, która zmienia emerge. Na kartce papieru znajduje się pamięć komputera i na tym papierze zapisywane są informacje - "Billy's Score" i "Jane's Score" to zmienne, miejsca w pamięci, w których przechowywane są wszystkie punkty gracza, a które zmieniają się z czasem.



W naszym przykładzie Scrabble zmienna ma dwa elementy - nazwę (np. "Wynik Jane") i wartość (np. 101). W przetwarzaniu zmienne mogą zawierać różne rodzaje wartości, a my musimy jednoznacznie zdefiniować rodzaj wartości, zanim będziemy mogli użyć danej zmiennej.

Ćwiczenie 4-1: Weź pod uwagę grę Pong. Jakie zmienne są potrzebne do zaprogramowania gry? (Jeśli nie znasz Ponga, zobacz http://pl.wikipedia.org/wiki/Pong).

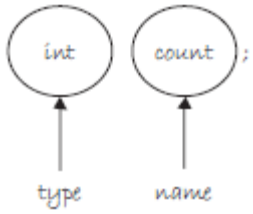
4.2 Deklaracja zmiennych i inicjalizacja

Zmienne mogą zawierać wartości pierwotne lub odniesienia do obiektów i tablic. Na razie będziemy tylko martwić się prymitywami - przejdziemy do obiektów i tablic w późniejszym rozdziale. Pierwotne wartości to budowane bloki danych na komputerze i zazwyczaj zawierają pojedynczą informację, taką jak liczba lub znak. Zmienne są deklarowane po pierwszym określeniu rodzaju, po którym następuje nazwa. Nazwy zmiennych muszą być jednym słowem (bez spacji) i muszą zaczynać się od litery (mogą zawierać liczby, ale nie mogą zaczynać się od cyfry). Nie mogą zawierać znaków interpunkcyjnych ani specjalnych, z wyjątkiem podkreślenia: "_". Typ jest rodzajem danych przechowywanych w tej zmiennej. Może to być liczba całkowita, liczba dziesiętna lub znak. Oto typ danych, z których będziesz często korzystać:

• Liczby całkowite, takie jak 0, 1, 2, 3, - 1, - 2 itd. Są przechowywane jako "liczby całkowite", a słowem kluczowym typu dla liczby całkowitej jest "int".

• Liczby dziesiętne, takie jak 3.14159, 2.5 i -9.95 są zwykle przechowywane jako "wartości zmiennoprzecinkowe",a słowem kluczowym typu zmiennoprzecinkowego jest "zmiennoprzecinkowe".

• Znaki, takie jak litery "a", "b", "c" itd. Są przechowywane w zmiennych typu "char" i są zadeklarowane jako litera ujęta w pojedyncze cudzysłowy, czyli "a". Znaki są przydatne przy ustalaniu jaka litera została naciśnięta na klawiaturze, i do innych zastosowań związanych z ciągami tekstu



Na powyższym rysunku mamy zmienną o nazwie "count" typu "int", która oznacza liczbę całkowitą. Inne możliwe typy danych są wymienione poniżej.

Nie zapomnij!

• Zmienne muszą mieć typ. Czemu? W ten sposób komputer dokładnie wie, ile pamięci należy przeznaczyć na przechowywanie danych tej zmiennej.

• Zmienne muszą mieć nazwę.

Wszystkie podstawowe typy

• boolean: true lub false

• char: postać, "a", "b", "c" itp.

• byte: mała liczba, od -128 do 127

• short: większa liczba od -32768 do 32767

• int: duża liczba, -2147483648 do 2147483647

• long: naprawdę ogromna liczba

• float: liczba dziesiętna, na przykład 3,14159

• double: liczba dziesiętna z dużo większą liczbą miejsc po przecinku (wymagana tylko w przypadku zaawansowanych programów wymagających dokładności matematycznej).

Po zadeklarowaniu zmiennej możemy ją przypisać wartością, ustawiając ją jako równą. W większości przypadków, jeśli zapomnimy zainicjować zmienną, przetwarzanie da jej wartość domyślną, na przykład 0 dla liczb całkowitych, 0.0 dla punktu flot i tak dalej. Jednak dobrze jest wejść w nawyk zawsze inicjowania zmiennych, aby uniknąć nieporozumień.

int count;

liczba = 50; : Deklaruje i inicjuje zmienną w dwóch wierszach kodu.

Aby być bardziej zwięzłym, możemy połączyć powyższe dwie wypowiedzi w jedną.

int count = 50; Deklaruje i inicjuje zmienną w jednym wierszu kodu.

Coś o nazwie?

Wskazówki dotyczące wyboru dobrych nazw zmiennych

• Unikaj używania słów, które pojawiają się w innym miejscu w języku Processing. Innymi słowy, nie dzwoń na zmienną mysz X, już istnieje!

• Używaj nazw, które coś znaczą. To może wydawać się oczywiste, ale jest to ważna kwestia. Na przykład, jeśli używasz zmiennej do śledzenia wyniku, nazwij ją "wynikiem", a nie powiedzmy "kot. "

• Uruchom zmienną małymi literami i łącz ze sobą wyrazy z dużymi literami. Słowa rozpoczynające się od wielkich liter są zarezerwowane dla klas. Na przykład: "frogColor" jest dobre, "Frogcolor" nie jest.

Zmienna może być również zainicjowana przez inną zmienną (x równa się y) lub przez oszacowanie wyrażenia matematycznego (x jest równe y plus z itd.). Oto kilka przykładów:

Przykład 4-1: Przykłady deklaracji zmiennych i inicjalizacji

```
int count = 0; // Zadeklaruj zmienna int nazwaną count której przypisano wartość 0

char letter = 'a'; // Zadeklaruj char o nazwie letter, któremu przypisano wartość "a"

double d = 132,32; // Deklaracja double o nazwie d, przypisanej wartości 132.32

boolean happy = false; // Zadeklaruj wartość boolean o nazwie happy, której przypisano wartość false

float x = 4,0; // Deklaracja float o nazwie x, której przypisano wartość 4.0

float y; // Zadeklaruj float o nazwie y (bez przydziału)

y = x + 5,2; // Przypisz wartość x plus 5.2 do poprzednio zadeklarowanej y

float z = x * y + 15,0; // Zadeklaruj zmienną o nazwie z, przypisz jej wartość, którą

// jest x razy y plus 15.0.
```

Ćwiczenie 4-2: Napisz deklarację zmiennych i inicjalizację gry Pong.

4.3 Używanie zmiennej

Być może początkowo wydaje się, że bardziej skomplikowane są słowa stojące w liczbach, zmienne sprawiają, że nasze życie jest łatwiejsze i bardziej interesujące. Zróbmy prosty przykład programu, który rysuje koło na ekranie.

Za chwilę dodamy tutaj zmienne u góry.

```
void setup() {

size(200,200);

}

void draw() {

background(255);

stroke(0);

fill(175);

ellipse(100,100,50,50);

}
```

Wcześniej dowiedzieliśmy się, jak wykonać ten prosty przykład o krok dalej, zmieniając położenie kształtu na mouseX, mouseY, aby przypisać jego położenie do myszy.

ellipse (mysz X, mysz Y, 50,50);

Czy widzisz, gdzie to się dzieje? mouseX i mouseY są nazwanymi odniesieniami do horyzontalnej i pionowej lokalizacji myszy. To zmienne! Jednakże, ponieważ są one wbudowane w środowisko przetwarzania (zauważ, że są one w kolorze czerwonym po wpisaniu ich w kodzie), mogą być używane bez deklaracji. Wbudowane zmienne (zmienne AKA "System") są omówione dalej w następnej sekcji. Teraz chcemy stworzyć własne zmienne, stosując składnię deklaracji i inicjalizacji przedstawioną powyżej, umieszczając zmienne u góry naszego kodu. Możesz zadeklarować zmienne w innym miejscu kodu, a my przejdziemy do tego później. Aby uniknąć zamieszania, wszystkie zmienne powinny być na górze.

Praktyczna Rada : Kiedy używać zmiennej

Nie ma sztywnych reguł, jeśli chodzi o użycie zmiennej. Jednakże, jeśli znajdziesz się w kilku liczbach podczas programowania, poświęć kilka minut, przejrzyj swój kod i zmień te wartości na zmienne. Niektórzy programiści twierdzą, że jeśli liczba pojawia się trzy lub więcej razy, powinna być zmienną. Osobiście powiedziałbym, że jeśli liczba pojawia się raz, użyj zmiennej. Zawsze używaj zmiennych!

Przykład 4-2: Używanie zmiennych

```
int circleX = 100;

int circleY = 100;  // Deklaruj i zainicjuj dwie zmienne liczb całkowitych u góry kodu.

void setup() {

size(200,200);

}

void draw() {

background(100);

stroke(255);

fill(0);

ellipse(circleX,circleY,50,50); // Użyj zmiennych, aby określić położenie elipsy.

}
```

Uruchamiając ten kod, uzyskujemy ten sam rezultat, co w pierwszym przykładzie: kółko pojawia się na środku ekranu. Niemniej jednak powinniśmy otworzyć nasze serca i przypomnieć sobie, że zmienna nie jest po prostu symbolem zastępczym dla jednej stałej wartości. Nazywamy to zmienną, ponieważ jest różna. Aby zmienić jego wartość, zapisujemy operację przypisania, która przypisuje nową wartość. Do tej pory każda linia kodu, którą napisaliśmy, nazywała się funkcją: line(), ellipse(), stroke(), etc. Zmienne wprowadzają operacje przypisania do miksu. Oto, jak wygląda jeden (to jest to samo, co sposób zainicjowania zmiennej, tylko zmienna nie musi być zadeklarowana).

nazwa zmiennej = wyrażenie

```
x = 5;

x = a + b;

x = y - 10 * 20;   // Przykłady przypisania nowej wartości do zmiennych.

x = x * 5;
```

Typowym przykładem jest inkrementacja. W powyższym kodzie koło X zaczyna się od wartości 100. Jeśli chcemy zwiększyć krąg X o jeden, mówimy, że circleX równa się sobie plus jeden. W kodzie oznacza to

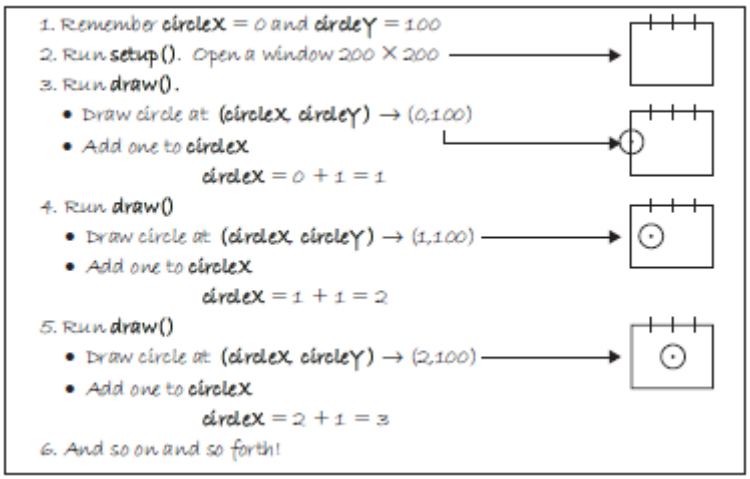
"circle X = circle X + 1; ".

Spróbujmy dodać to do naszego programu (i zacznijmy od kręguX o wartości 0).

Przykład 4-3: Różne zmienne

```
int circle X = 0;

int circleY = 100;

void setup() {

size(200,200);

}

void draw() {

background(255);

stroke(0);

fill(175);

ellipse(circleX,circleY,50,50);

circleX = circleX + 1;    //  Operacja przypisania, która zwiększa wartość circleX o 1.

}
```

Co się dzieje? Jeśli uruchomisz Przykład 4-3 w Obróbce, zauważysz, że kółko porusza się od lewej do prawej. Pamiętaj, że ciągnij () pętle w kółko, cały czas zachowując wartość circleX w pamięci. Udawajmy, że jesteśmy przez chwilę komputerem. (To może wydawać się zbyt proste i oczywiste, ale jest kluczem do naszego zrozumienia zasad programowania ruchu.)



Ćwiczenie, w jaki sposób postępować zgodnie z kodem krok po kroku, prowadzi do pytań, które należy zadać przed napisaniem własnych szkiców. Bądź jednym z komputerem.

• Jakie dane, które Ty i Twój komputer powinniście zapamiętać dla swojego szkicu?

• W jaki sposób ty i komputer używasz tych danych do rysowania kształtów na ekranie?

• W jaki sposób ty i komputer zmieniasz dane, aby uczynić szkic interaktywnym i animowanym?

Ćwiczenie 4-3: Zmień przykład 4-3, aby zamiast okręgu poruszającego się od lewej do prawej, okrąg powiększył się. Co byś zmienił, aby kółko podążało za myszą w miarę wzrostu? Jak możesz zmieniać szybkość, z jaką krąg się powiększa?

```
int circleSize = 0;

int circleX = 100;

int circleY = 100;

void setup() {

size(200,200);

}

void draw() {

background(0);

stroke(255);

fill(175);

_____

_____

}
```

4.4 Wiele zmiennych

Przyjrzyjmy się przykładowi dalej i wykorzystajmy zmienne dla każdej informacji, którą możemy wymyślić. Użyjemy również wartości punktów flotacyjnych, aby wykazać większą precyzję w dostosowywaniu wartości zmiennych.

Przykład 4-4: Wiele zmiennych

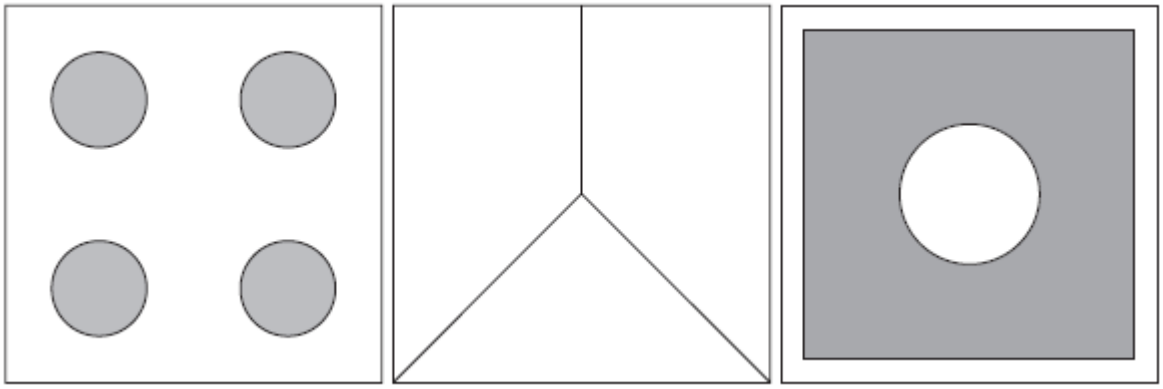
```
float circleX = 0;

float circleY = 0;   // Mamy teraz osiem zmiennych! Wszystkie typy float

float circleW = 50;

float circleH = 100;

float circleStroke = 255;

float circleFill = 0;

float backgroundColor = 255;

float change = 0.5;

// Your basic setup

void setup() {

size(200,200);

smooth();

}

void draw() {

// Draw the background and the ellipse

background(backgroundColor);   // Zmienne są używane do wszystkiego: tła, obrysu, wypełnienia,
                                //lokalizacji i rozmiaru.

stroke(circleStroke);

fill(circleFill);

ellipse(circleX,circleY,circleW,circleH);

// Change the values of all variables

circleX = circleX + change;  //   Zmienna zmienna służy do zwiększania i zmniejszania innych zmiennych

circleY = circleY + change;

circleW = circleW + change;

circleH = circleH - change;

circleStroke = circleStroke - change;

circleFill = circleFill + change;

}
```

Ćwiczenie 4-4

Krok 1: Napisz kod, który rysuje następujące zrzuty ekranu z zakodowanymi wartościami. (Możesz używać kolorów zamiast skali szarości).

Krok 2: Zastąp wszystkie zakodowane numery zmiennymi.

Krok 3: Zapisz operacje przypisania w funkcji draw (), które zmieniają wartość zmiennych. Na przykład "zmienna1 = zmienna1 + 2; ". Wypróbuj różne wyrażenia i zobacz, co się stanie!



4.5 Zmienne systemowe

Jak widzieliśmy przy pomocy mouseX i mouseY, Processing ma swobodny dostęp do zestawu wygodnych zmiennych systemowych. Są to zwykle potrzebne dane powiązane ze wszystkimi szkicami (takie jak szerokość okna, klawisz naciśnięty na klawiaturze itp.). Podczas nazywania własnych zmiennych najlepiej unikać nazw zmiennych systemowych, jednak jeśli przypadkowo użyjesz ich, zmienna stanie się podstawowa i zastąpi systemową. Oto lista najczęściej używanych zmiennych systemowych (jest ich więcej, które można znaleźć w odnośniku Przetwarzanie).

• width -Width (w pikselach) okna szkicu.

• height-Wysokość (w pikselach) okna szkicu.

• frameCount - liczba przetworzonych ramek.

• frameRate - Wskaż, że ramki są przetwarzane (na sekundę).

• screen.width -width (w pikselach) całego ekranu.

• screen.height - Wysokość (w pikselach) całego ekranu.

• klawisz - Największy ostatnio naciskany klawisz na klawiaturze.

• keyCode - numeryczny kod klawisza wciśniętego na klawiaturze.

• keyPressed - Prawda czy fałsz? Czy naciśnięty jest klawisz?

• mousePressed -True lub false? Czy mysz jest wciśnięta?

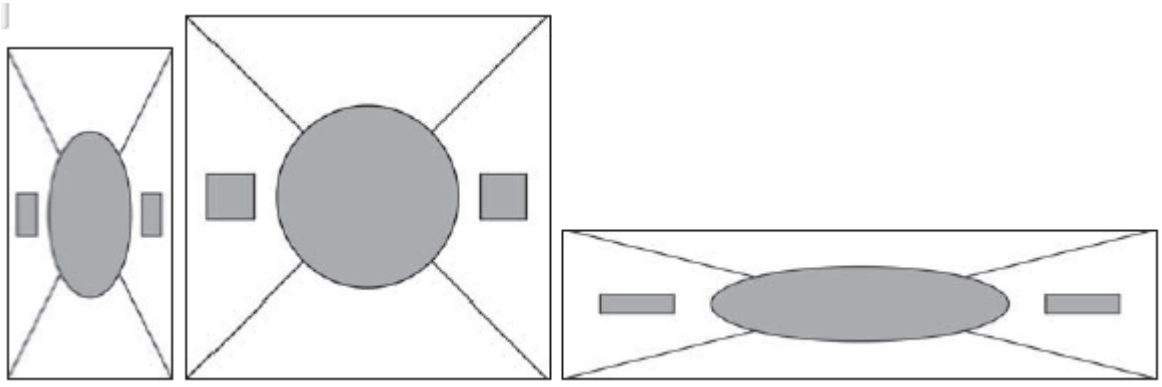
• mouseButton - Który przycisk jest wciśnięty? W lewo, w prawo lub w centrum?

Poniżej znajduje się przykład, który wykorzystuje niektóre z powyższych zmiennych; nie jesteśmy jeszcze gotowi, aby je wszystkie wykorzystać, ponieważ będziemy potrzebować bardziej zaawansowanych koncepcji, aby móc korzystać z wielu funkcji.

Przykład 4-5: Używanie zmiennych systemowych

```
void setup() {
size(200,200);
frameRate(30);
}
void draw() {
background(100);
stroke(255);
fill(frameCount/2);  // frameCount służy do pokolorowania prostokąta
rectMode(CENTER);
rect(width/2,height/2,mouseX + 10,mouseY + 10); // Prostokąt zawsze będzie znajdował się w środku
                                                //okna, jeśli znajduje się w (szerokość / 2, wysokość
                                                // / 2).
}
void keyPressed() {
println(key);
}
```

Ćwiczenie 4-5: Wykorzystując szerokość i wysokość, odtwórz poniższy zrzut ekranu. Oto haczyk: kształty muszą się zmieniać w stosunku do rozmiaru okna. (Innymi słowy, niezależnie od tego, co określisz dla size (), wynik powinien wyglądać identycznie.)



4.6 Losowo: różnorodność jest przyprawą życia.

Tak więc, być może zauważyłeś, że przykłady w tej książce do tej pory są trochę, powiedzmy, nudne. Koło tutaj. Kwadrat tutaj. Szarawy kolor. Kolejny szarawy kolor. Istnieje metoda na szaleństwo (lub brak szaleństwa w tym przypadku). Wszystko to wraca do zasady prowadzenia tej książki: stopniowego rozwoju. O wiele łatwiej jest nauczyć się podstaw, patrząc na poszczególne części, programy, które wykonują jedną i tylko jedną rzecz. Następnie możemy zacząć dodawać funkcje na górze, krok po kroku. Mimo to czekaliśmy cierpliwie na cztery rozdziały i doszliśmy do momentu, w którym możemy

zacząć się trochę zabawić. A ta zabawa zostanie zademonstrowana za pomocą funkcji random (). Rozważmy na chwilę przykład 4-6, którego wyniki pokazano na rysunku

Przykład 4-6: Elipsa ze zmiennymi

```
float r = 100;  // Zadeklaruj i zainicjuj swoje zmienne

float g = 150;

float b = 200;

float a = 200;

float diam = 20;

float x = 100;

float y = 100;

void setup() {

size(200,200);

background(255);

smooth();

}

void draw() {

// Use those variables to draw an ellipse

stroke(0);

fill(r,g,b,a);        //  Użyj tych zmiennych! (Pamiętaj, że czwartym argumentem dla koloru jest
                      //przezroczystość).

ellipse(x,y,diam,diam);

}
```

Tak jest, nasz ponury krąg. Oczywiście, możemy dostosować wartości zmiennych i przesunąć kółko, zwiększyć jego rozmiar, zmienić jego kolor i tak dalej. Co jednak, jeśli za każdym razem, poprzez draw (), możemy stworzyć nowy krąg, z losowym rozmiarem, kolorem i pozycją? Funkcja losowa () pozwala nam to zrobić dokładnie. random () jest specjalnym rodzajem funkcji, jest funkcją, która zwraca wartość. Napotkaliśmy to już wcześniej. W ćwiczeniu 3-7 użyliśmy funkcji abs () do obliczenia wartości bezwzględnej liczby. Pomysł funkcji, która oblicza wartość i zwraca ją, zostanie w pełni zbadany w części 7, ale teraz zajmie nam trochę czasu, aby wprowadzić pomysł i pozwolić mu na chwilę. W

przeciwieństwie do większości funkcji, z którymi mamy do czynienia (np. Line (), ellipse () i rect ()), random () nie rysuje ani nie koryguje kształtu na ekranie. Zamiast tego random () odpowiada na pytanie; zwraca nam tę odpowiedź. Oto trochę dialogu. Zapraszam do próby z przyjaciółmi.

Ja: Hej Losowy, co się dzieje? Mam nadzieje, że masz się dobrze. Posłuchaj, zastanawiałem się, czy mógłbyś dać mi liczbę losową od 1 do 100?

Losowy: jak, bez problemu. A co z liczbą 63?

Ja: Jestem niesamowity, naprawdę świetny, dziękuję. OK, mam wolne. Muszę narysować prostokąt o szerokości 63 pikseli, dobrze?

Jak wyglądałaby ta sekwencja w naszym nieco bardziej formalnym środowisku przetwarzania? Kod poniżej części "ja" jest odtwarzany przez zmienną "w".

```
float w = losowy (1100);  //  Losowy float od 1 do 100.

rect (100, 100, w, 50);
```

Funkcja random() wymaga dwóch argumentów i zwraca losową liczbę punktów od pierwszego do drugiego argumentu. Drugi argument musi być większy niż pierwszy, aby działał prawidłowo. Funkcja random() działa również z jednym argumentem, przyjmując zakres od zera do tego argumentu. Ponadto random() zwraca tylko numery punktów flotacyjnych. Właśnie dlatego zadeklarowaliśmy "w" powyżej jako flot. Jeśli jednak chcesz losową liczbę całkowitą, możesz przekonwertować wynik funkcji losowej na int.

```
int w = int (losowy (1100));  //  Losowa liczba całkowita od 1 do 100.

rect (100, 100, w, 50);
```

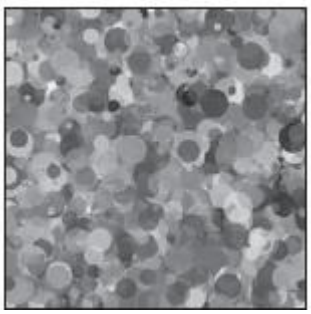
Zwróć uwagę na użycie zagnieżdżonych nawiasów. Jest to miła koncepcja, do której należy się przyzwyczaić, ponieważ wygodnie będzie wywoływać funkcje wewnątrz funkcji, jak to robimy. Funkcja losowa () zwraca strumień, który następnie jest przekazywany do funkcji int (), która konwertuje ją na liczbę całkowitą. Gdybyśmy chcieli użyć funkcji zagnieżdżania, moglibyśmy nawet skondensować powyższy kod w jedną linię:

```
rect(100,100,int(random(1,100)),50);
```

Nawiasem mówiąc, proces konwersji jednego typu danych na inny jest określany jako "rzutowanie. "W języku Java (na którym oparte jest przetwarzanie) rzutowanie pliku pływającego na liczbę całkowitą można również zapisać w następujący sposób:

```
int w = (int) random(1,100);      // Wynikiem losowym (1100) jest flop. Można go przekonwertować na
                                  //liczbę całkowitą przez "rzutowanie.
```

OK, jesteśmy teraz gotowi na eksperymenty z random (). Przykład 4-7 pokazuje, co się stanie, jeśli weźmiemy każdą zmienną związaną z rysowaniem elipsy (fi ll, położenie, rozmiar) i przypiszemy ją do liczby losowej każdego cyklu przez draw (). Dane wyjściowe pokazano na rysunku



```
float r;
float g;
float b;
float a;
float diam;
float x;
float y;
void setup() {
size(200,200);
background(0);
smooth();
}
void draw() {
// Fill all variables with random values
r = random(255);
g = random(255);
b = random(255);        //Za każdym razem poprzez draw () wybierane są nowe losowe liczby dla nowej
                        //elipsy.
a = random(255);
diam = random(20);
x = random(width);
y = random(height);
// Use values to draw an ellipse
noStroke();
fill(r,g,b,a);
```

```
ellipse(x,y,diam,diam);

}
```

4.7 Zmienne Stworka

Jesteśmy teraz gotowi, aby ponownie odwiedzić Zooga, naszego przyjaciela z kosmosu, który z radością podążał za myszą po ekranie, kiedy ostatni raz się zameldowaliśmy. Tutaj dodamy dwa elementy funkcjonalności do Zooga.

• Nowa funkcja # 1 -Zoog wzrośnie spod ekranu i odleci w przestrzeń kosmiczną (nad ekranem).

• Nowa funkcja # 2 - Oczy Zooga będą losowo kolorowane w miarę poruszania się Zooga.

Funkcja nr 1 jest rozwiązywana przez zwykłe użycie poprzedniego programu, który używał mouseX i mouseY i zastępowanie własnych zmiennych w ich miejscu.

Funkcja # 2 jest zaimplementowana poprzez utworzenie trzech dodatkowych zmiennych eyeRed, eyeGreen i eyeBlue, które będą używane do funkcji fi ll () przed wyświetleniem elips ocznych.

Przykład 4-8: Zmienne Stworka

```
float zoogX;    //Deklarowanie zmiennych. zoogX i zoogY są dla funkcji # 1. eyeR, eyeG, eyeB są dla
                // funkcji # 2.

float zoogY;

float eyeR;

float eyeG;

float eyeB;

void setup() {

size(200,200);

zoogX = width/2; // Zoog always starts in the middle

zoogY = height + 100; // Zoog starts below the screen

smooth();       // Funkcja nr 1. zoogX i zoogY są inicjowane w zależności od wielkości okna. Zauważ,
                //że nie możemy zainicjować tych zmiennych przed wywołaniem funkcji size (),
                //ponieważ używamy wbudowanych zmiennych szerokości i wysokości.

}

void draw() {

background(255);

// Set ellipses and rects to CENTER mode

ellipseMode(CENTER);

rectMode(CENTER);

// Draw Zoog's body

stroke(0);
```

```
fill(150);

rect(zoogX,zoogY,20,100); //Funkcja nr 1. zoogX i zoogY są używane do lokalizacji kształtów.

// Draw Zoog's head

stroke(0);

fill(255);

ellipse(zoogX,zoogY - 30,60,60);

// Draw Zoog's eyes

eyeR = random(255);    // unkcja nr 2. eyeR, eyeG i eye B otrzymują wartości losowe i są używane w
                       //funkcji fill ().

eyeG = random(255);

eyeB = random(255);

fill(eyeR,eyeG,eyeB);

ellipse(zoogX - 19,zoogY - 30,16,32);

ellipse(zoogX + 19,zoogY - 30,16,32);

// Draw Zoog's legs

stroke(150);

line(zoogX - 10,zoogY + 50,zoogX - 10,height);

line(zoogX + 10,zoogY + 50,zoogX + 10,height);

// Zoog moves up

zoogY = zoogY - 1;     // Funkcja nr 1. zoogY zmniejsza się o jeden, aby zoog przesuwał się w górę na
                       //ekranie

}
```

Ćwiczenie 4-6: Zmień Przykład 4-8, aby Zoog poruszał się w lewo i prawo, gdy Zoog porusza się w górę. Wskazówka: wymaga to użycia random () w połączeniu z zoogX.

```
zoogX = _____;
```

Ćwiczenie 4-7: Używając zmiennych i funkcji losowej (), zmień swój projekt z Projektu lekcji 1, aby poruszać się po ekranie, zmieniać kolor, rozmiar, lokalizację i tak dalej.

V WARUNKOWOŚĆ

5.1 Wyrażenia logiczne

Jaki jest twój ulubiony rodzaj testu? Format eseju? Wielokrotny wybór? W świecie programowania komputerowego bierzemy tylko jeden rodzaj testu: test boolowski - prawda lub fałsz. Wyrażenie boolowskie (nazwane od matematyka George'a Boole'a) jest wyrażeniem, które zwraca wartość prawda lub fałsz. Spójrzmy na kilka popularnych przykładów językowych:

* Jestem głodny → prawda

* Obawiam się programowania komputerowego. → false

* Ta książka to zabawna lektura. → false

W formalnej logice informatyki testujemy relacje między liczbami.

* 15 jest większy niż 20 → fałsz

* 5 równa się 5 → prawda

* 32 jest mniejszy lub równy 33 → true

Tu nauczymy się używać zmiennej w wyrażeniu boolowskim, dzięki czemu nasz szkic może przyjmować różne ścieżki w zależności od aktualnej wartości przechowywanej w zmiennej.

* x > 20 → zależy od bieżącej wartości x

* y = = 5 → zależy od aktualnej wartości y

* z < =33 → zależy od aktualnej wartości z

Następujące operatory mogą być używane w wyrażeniu boolowskim.

Operatory relacyjne

> większy niż

< = mniejszy lub równy

< mniejszy niż

= = równość

> = większy niż lub równy

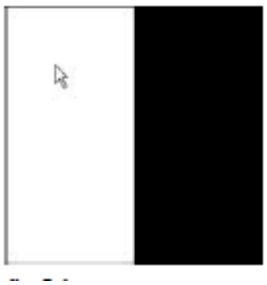
!= nierówność

5.2 Warunki: Jeśli, Else, Else If

Wyrażenia logiczne (często określane jako "warunkowe") działają w obrębie szkicu jako pytania. Czy 15 jest większe niż 20? Jeśli odpowiedź brzmi "tak" (tj. "Prawda"), możemy wybrać wykonanie pewnych instrukcji (np. narysuj prostokąt); jeśli odpowiedź brzmi "nie" (tj. "fałsz"), instrukcje te są ignorowane. Wprowadza ideę rozgałęzienia; w zależności od różnych warunków program może śledzić różne ścieżki. W świecie fizycznym może to oznaczać takie instrukcje: Jeśli jestem głodny, to zjem jakieś jedzenie, inaczej jeśli jestem spragniony, napij się wody, w przeciwnym razie zdrzemnij się. W przetwarzaniu możemy mieć coś więcej jak:

Jeśli mysz znajduje się po lewej stronie ekranu, narysuj prostokąt po lewej stronie ekranu. Lub, bardziej formalnie, z wynikami pokazanymi na rysunku

```
if (mouseX <width / 2) {

wypełnienie (255);

rect (0,0, szerokość / 2, wysokość);

}
```



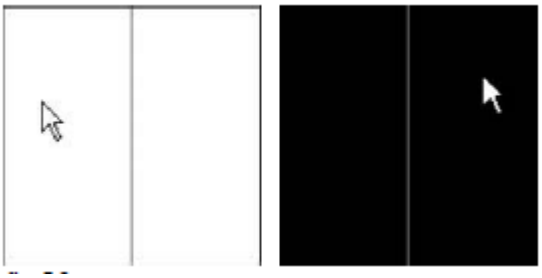
Wyrażenie boolowskie i instrukcje wynikające z powyższego kodu źródłowego są zawarte w bloku kodu o następującej składni i strukturze:

```
if (wyrażenie logiczne) {

// kod do wykonania, jeśli wyrażenie boolean ma wartość true

}
```

Strukturę można rozszerzyć za pomocą słowa kluczowego else, aby uwzględnić kod, który jest wykonywany, jeśli wyrażenie boolowskie jest fałszywe. Th jest odpowiednikiem "w przeciwnym razie, czyń takie i takie. "

```
if (wyrażenie logiczne) {

// kod do wykonania, jeśli wyrażenie boolean ma wartość true

} else {

// kod do wykonania, jeśli wyrażenie boolean ma wartość false

}
```

Na przykład możemy powiedzieć, co następuje, z wyjściem pokazanym na rysunku.



Jeśli mysz znajduje się po lewej stronie ekranu, narysuj białe tło, w przeciwnym razie narysuj czarne tło.

if (mouseX <width / 2) {

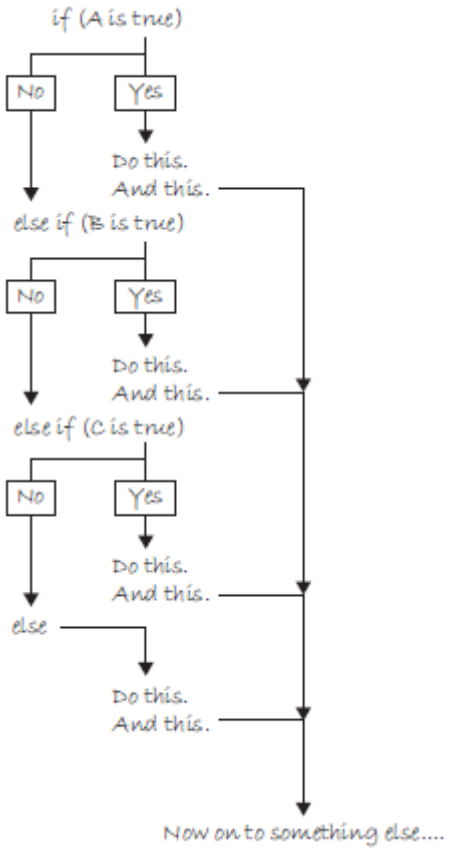
tło (255);

} else {

tło (0);

}

Na koniec, aby przetestować wiele warunków, możemy użyć "else if. "Jeśli użyto else, wyrażenia warunkowe są oceniane w przedstawionej kolejności. Gdy tylko jedno wyrażenie logiczne zostanie uznane za prawdziwe, odpowiedni kod zostanie wykonany, a pozostały Wyrażenia boolowskie są ignorowane. Zobacz rysunek



if (wyrażenie boolean # 1) {

// kod do wykonania, jeśli wyrażenie logiczne # 1 ma wartość true

} else if (boolean expression # 2) {

// kod do wykonania, jeśli wyrażenie logiczne # 2 jest prawdziwe

} else if (boolean expression #n) {

// kod do wykonania, jeśli wyrażenie logiczne # n ma wartość true

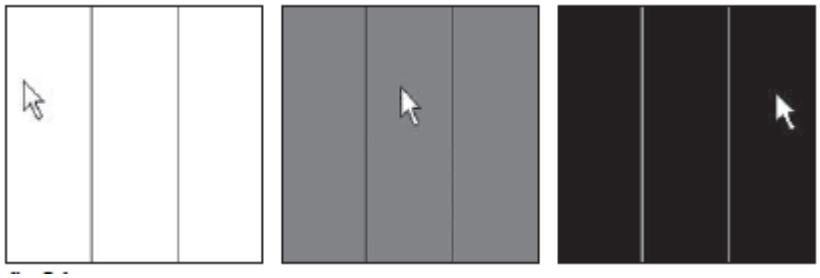
} else {

// kod do wykonania, jeśli żaden z powyższych

// Wyrażenia boolean są prawdziwe

}

Biorąc nasz prosty przykład myszy o krok dalej, moglibyśmy powiedzieć następnie, z wynikami pokazanymi na rysunku.



Jeśli mysz znajduje się w lewej trzeciej części okna, narysuj białe tło, jeśli znajduje się w środkowej trzeciej części, narysuj szare tło, w przeciwnym razie narysuj czarne tło.

if (mouseX <width / 3) {

background (255);

} else if (mouseX <2 * width / 3) {

tło (127);

} else {

background (0);

}

Ćwiczenie 5-1: Rozważmy system klasyfikacji, w którym liczby są zamieniane na litery. Wypełnij puste pola poniższym kodem, aby zakończyć wyrażenie boolowskie.

float grade = random(0,100);

if (_____) {

println("Assign letter grade A. ");

} else if (_____) {

println (_____);

} else if (_____) {

println(_____);

} else if (_____) {

println(_____);

} else {

println(_____);

}

Ćwiczenie 5-2: Sprawdź poniższe próbki kodu i określ, co pojawi się w oknie komunikatu. Zapisz odpowiedź, a następnie wykonaj kod w Processing, aby porównać.

Problem nr 1: Ustal, czy liczba mieści się w przedziale od 0 do 25, od 26 do 50 lub od 50.

```
int x = 75;

if (x > 50) {

println(x + " is greater than

50! " );

} else if (x > 25) {

println(x + " is greater than

25! " );

} else {

println(x + " is 25 or

less! " );

}
```

WYJŚCIE ----------

```
int x = 75;

if(x > 25) {

println(x + " is greater

than 25! " );

} else if (x > 50) {

println(x + " is greater

than 50! " );

} else {

println(x + " is 25 or

less! " );

}
```

WYJŚCIE ----------

Mimo że składnia jest poprawna, co jest problematycznego w odniesieniu do kodu z pierwszej kolumny?

Warto zauważyć, że w Ćwiczeniu 5-2, kiedy testujemy równość, musimy użyć dwóch równych znaków. Dzieje się tak dlatego, że przy programowaniu pytanie, czy coś jest równe, różni się od przypisania wartości do zmiennej.

if (x = = y) { // "Czy x jest równe y?" Użyj podwójnej równości!

x = y; // "Ustaw x równy y." Użyj pojedynczego równego!

5.3 Wyrażenia warunkowe w szkicu

Spójrzmy na bardzo prosty przykład programu, który wykonuje różne zadania w oparciu o wynik pewnych warunków. Nasz pseudokod jest poniżej.

Krok 1. Utwórz zmienne, aby przytrzymać składniki koloru czerwonego, zielonego i niebieskiego. Nazwijmy je r, g i b.

Krok 2. Ciągłe rysowanie tła w oparciu o te kolory.

Krok 3. Jeśli mysz znajduje się po prawej stronie ekranu, zwiększ wartość r, jeśli jest po lewej stronie, zmniejsz wartość r.

Krok 4. Ogranicz wartość r do 0 i 255.

Th jest pseudokod jest zaimplementowany w Processing w przykładzie 5-1.

Przykład 5-1:

```
float r = 150;

float g = 0;   //    Zmienne

float b = 0;

void setup() {

size(200,200);

}

void draw() {    // Rysuj rzeczy

background(r,g,b);

stroke(255);

line(width/2,0,width/2,height);

if(mouseX > width/2) {              // "Jeśli mysz znajduje się po prawej stronie ekranu" jest
                                    //równoważne "jeśli mouseX jest większy niż szerokość podzielona
                                    //przez 2."

r = r + 1;

} else {

r = r - 1;

}

if (r > 255) {  // Jeśli r jest większe niż 255, ustaw je na 255. Jeśli r jest mniejsze od 0, ustaw je na 0.
```

```
r = 255;

} else if (r < 0) {

r = 0;

}

}
```

Ograniczanie wartości zmiennej, jak w kroku 4, jest częstym problemem. Tutaj nie chcemy, aby wartości kolorów wzrastały do nieuzasadnionych ekstremów. W innych przykładach możemy chcieć ograniczyć rozmiar lub umiejscowienie kształtu, aby nie stał się zbyt duży lub zbyt mały, ani nie oddalił się od ekranu. Podczas gdy użycie instrukcji if jest całkowicie poprawnym rozwiązaniem problemu z ograniczeniem, przetwarzanie kończy się funkcją zwaną constrain (), która dostarczy nam ten sam wynik w jednym wierszu kodu.

```
if (r > 255) { // Ograniczaj za pomocą instrukcji "if"

r = 255;

} else if (r < 0) {

r = 0;

}

r = constrain(r,0,255);  // Ograniczaj za pomocą funkcji constrain()
```

constrain () pobiera trzy argumenty: wartość, którą zamierzamy ograniczyć, minimalny limit i maksymalny limit. Funkcja ta zwraca "ograniczoną" wartość i jest przypisana do zmiennej r. (Zapamiętaj, co to znaczy, że funkcja zwraca wartość? Zobacz naszą dyskusję na temat random ().)Nabieranie nawyku ograniczania wartości to świetny sposób na uniknięcie błędów; bez względu na to, na ile jesteś pewien, że twoje zmienne pozostaną w danym zakresie, nie ma żadnych gwarancji poza samym ograniczeniem (). A pewnego dnia, podczas pracy nad większymi projektami programowymi z wieloma programistami, funkcje takie jak constrain () mogą zapewnić, że sekcje kodu dobrze ze sobą współpracują. Obsługa błędów, zanim się pojawią Kod jest symbolem dobrego stylu. Zróbmy nasz pierwszy przykład nieco bardziej zaawansowany i zmień wszystkie trzy komponenty koloru zgodnie z lokalizacja myszy i stan kliknięcia. Zwróć uwagę na użycie funkcji constrain () dla wszystkich trzech zmiennych. Zmienna systemowa mousePressed ma wartość true lub false w zależności od tego, czy użytkownik przytrzymuje przycisk myszy.

Przykład 5-2: Więcej warunków

```
float r = 0;

float b = 0;

float g = 0; //  Trzy zmienne dla koloru tła.

void setup() {

size(200,200);

}

void draw() {
```

```
background(r,g,b);  // Pokoloruj tło i narysuj linie, aby podzielić okno na ćwiartki.

stroke(0);

line(width/2,0,width/2,height);

line(0,height/2,width,height/2);

if(mouseX > width/2) {  // Jeśli mysz znajduje się po prawej stronie okna, zwiększ ją na czerwono. W
                        //przeciwnym razie jest po lewej stronie i zmniejsza się na czerwono

r = r + 1;

} else {

r = r - 1;

}

if (mouseY > height/2) { // Jeśli mysz znajduje się w dolnej części okna, zwiększ ją na niebiesko. W
                         //przeciwnym razie jest on na górze i zmniejsza się na niebiesko.

b = b + 1;

} else {

b = b - 1;

}

if (mousePressed) {      // Jeśli mysz zostanie naciśnięta (przy użyciu zmiennej systemowej
                         //mousePressed), należy zwiększyć kolor na zielony.

g = g + 1;

} else {

g = g - 1;

}

r = constrain(r,0,255);

g = constrain(g,0,255); // Ogranicz wszystkie wartości kolorów do wartości od 0 do 255.

b = constrain(b,0,255);

}
```

Ćwiczenie 5-3: Przesuń prostokąt przez okno, zwiększając wartość zmiennej. Rozpocznij kształt przy współrzędnej x 0 i użyj instrukcji if, aby zatrzymać ją na współrzędnej 100. Przepisz szkic, aby użyć constrain () zamiast instrukcji if. Uzupełnij brakujący kod.

```
// Prostokąt rozpoczyna się w lokalizacji x

float x = 0;

void setup() {
```

```
size(200,200);
}
void draw() {
background(255);
// Display object
fill(0);
rect(x,100,20,20);
// Increment x
x = x + 1;
_____
_____
_____
}
```

5.4 Operatory logiczne

Podbiliśmy proste stwierdzenie:

Jeśli moja temperatura jest większa niż 98,6, weź mnie do lekarza. Czasami jednak samo wykonanie zadania w oparciu o jeden warunek nie wystarczy. Na przykład:

Jeśli moja temperatura jest większa niż 98,6 LUB mam wysypkę na ramieniu, weź mnie do lekarza.

Jeśli ukąsi mnie pszczoła I mam alergię na pszczoły, zabierz mnie do lekarza.

Będziemy często robić to samo w programowaniu od czasu do czasu.

Jeśli mysz znajduje się po prawej stronie ekranu ORAZ gdy mysz znajduje się u dołu ekranu, narysuj prostokąt w prawym dolnym rogu.

Naszym pierwszym instynktem może być napisanie powyższego kodu za pomocą zagnieżdżonej instrukcji if, na przykład:

```
if (mouseX > width/2) {
if (mouseY > height/2) {
fill(255);
rect(width/2,height/2,width/2,height/2);
}
}
```

Innymi słowy, musielibyśmy ominąć dwie instrukcje if, zanim osiągniemy kod, który chcemy wykonać. To działa, owszem, ale można to zrobić w prostszy sposób, używając tego, co nazywamy "logicznym", napisanym jako dwa znaki handlowe ("&"). Pojedynczy znak ampersand ("&") oznacza coś innego w

przetwarzaniu, więc upewnij się, że uwzględnisz dwa! "Logiczny" lub "to dwa pionowe paski (AKA dwa" przewody ")" || ". Jeśli nie możesz znaleźć rury, zwykle jest ona na klawiaturze jako ukośnik odwrotny.

if (mouseX > width/2 & & mouseY > height/2) {

fill(255); // Jeśli mysz znajduje się po prawej stronie i na dole.

rect(width/2,height/2,width/2,height/2);

}

Oprócz & & i || mamy również dostęp do operatora logicznego "nie", napisanego jako wykrzyknik:!

Jeśli moja temperatura NIE jest większa niż 98,6, nie wezmę chorego do pracy.

Jeśli jestem użądlony przez pszczołę I NIE jestem uczulony na pszczoły, nie martw się!

Przykład przetwarzania to:

Jeśli mysz NIE jest naciśnięta, narysuj okrąg, w przeciwnym razie narysuj kwadrat.

```
if (!mousePressed) {    // ! znaczy nie. "MousePressed" jest zmienną typu boolean, która działa jako
                        //własne wyrażenie logiczne. Jego wartość to prawda lub fałsz (w zależności
                        //od tego, czy mysz jest aktualnie naciśnięta).
```

ellipse(width/2,height/2,100,100);

} else {

rect(width/2,height/2,100,100);

}

Zauważ, że ten przykład może być napisany z pominięciem, mówiąc:

Jeśli mysz zostanie naciśnięta, narysuj kwadrat, w przeciwnym razie narysuj okrąg.

Ćwiczenie 5-4: Czy następujące wyrażenia boolean są prawdziwe lub fałszywe? Załóżmy zmienne

x = 5 i y = 6.

!(x > 6) _____ _____

(x = = 6 & & x = = 5)_____

(x = = 6 || x = = 5)_____

(x > -1 & & y < 10)_____ _____

Chociaż składnia jest poprawna, co jest wadliwe w odniesieniu do następującego wyrażenia boolowskiego?

(x > 10 & & x < 5) _____

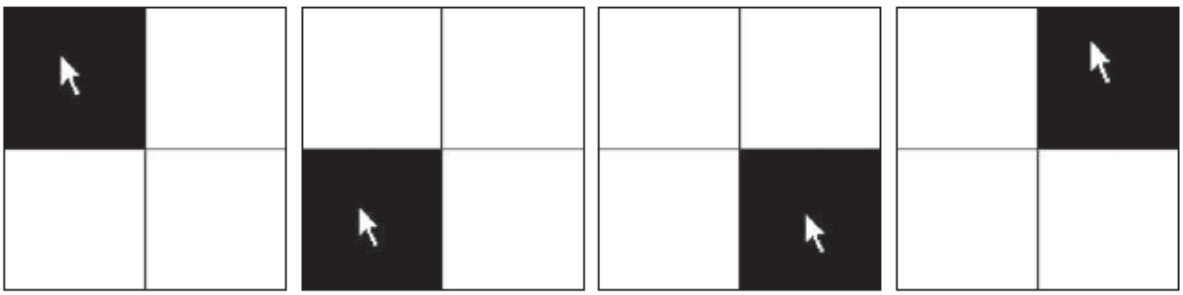
Ćwiczenie 5-5: Napisz program realizujący prosty najazd. Innymi słowy, jeśli mysz znajduje się nad prostokątem, prostokąt zmienia kolor. Oto kod, który pomoże Ci zacząć

int x = 50;

```
int y = 50;

int w = 100;

int h = 75;

void setup() {

size(200,200);

}

void draw() {

background(0);

stroke(255);

if (_______ & & _______ & & _______ & & _______) {

______

} ______ {

_______

}

rect(x,y,w,h);

}
```

5.5 Wielokrotne najazdy

Rozwiążmy prosty problem, nieco bardziej zaawansowaną wersję Ćwiczenia 5-5. Rozważ cztery zrzuty ekranu pokazane na rysunku z jednego szkicu. Biały kwadrat jest wyświetlany w jednym z czterech kwadrantów, zgodnie z lokalizacją myszy.



Najpierw zapiszmy logikę naszego programu w pseudokodach (to znaczy po angielsku).

Ustawienie

1. Ustaw okno o rozdzielczości 200 200 pikseli.

Rysowanie:

1. Narysuj białe tło.

2. Narysuj linie poziome i pionowe, aby podzielić okno na cztery ćwiartki.

3. Jeśli mysz znajduje się w lewym górnym rogu, narysuj czarny prostokąt w lewym górnym rogu.

4. Jeśli mysz znajduje się w prawym górnym rogu, narysuj czarny prostokąt w prawym górnym rogu.

5. Jeśli mysz znajduje się w lewym dolnym rogu, narysuj czarny prostokąt w lewym dolnym rogu.

6. Jeśli mysz znajduje się w prawym dolnym rogu, narysuj czarny prostokąt w prawym dolnym rogu.

W przypadku instrukcji od 3 do 6 musimy zadać sobie pytanie: - Skąd wiemy, czy mysz znajduje się w danym rogu? Aby to osiągnąć, musimy opracować bardziej szczegółowe instrukcje if. Na przykład, powiedzielibyśmy: - Jeśli lokalizacja myszy myszy jest większa niż 100 pikseli, a lokalizacja myszy Y jest większa niż 100 pikseli, narysuj czarny prostokąt w prawym dolnym rogu. Jako ćwiczenie możesz spróbować napisać ten program samodzielnie na podstawie powyższego pseudokodu. Odpowiedź, którą podajemy, podano w przykładzie 5-3.

Przykład 5-3: Najazdy

```
void setup() {
size(200,200);
}
void draw() {
background(255);
stroke(0);
line(100,0,100,200);
line(0,100,200,100);
// Fill a black color
noStroke();
fill(0);
if (mouseX < 100 & & mouseY < 100) {
rect(0,0,100,100);
} else if (mouseX > 100 & & mouseY < 100) {
rect(100,0,100,100); // W zależności od położenia myszy wyświetlany jest inny prostokąt.
} else if (mouseX < 100 & & mouseY > 100) {
rect(0,100,100,100);
} else if (mouseX > 100 & & mouseY > 100) {
rect(100,100,100,100);
}
}
```

Ćwiczenie 5-6: Przepisz Przykład 5-3, aby kwadraty zmieniały się z białego na czarny, gdy mysz opuścił ich obszar. Wskazówka: potrzebujesz czterech zmiennych, po jednej dla każdego koloru prostokąta.

5.6 Zmienne typu Boolean

Naturalnym krokiem naprzód od programowania rolowania jest przycisk. W końcu przycisk jest tylko rolowaniem, które reaguje po kliknięciu. Teraz może być nieco rozczarowujące programowanie najazdów i przycisków. Być może myślisz: "Czy nie mogę po prostu wybrać" Dodaj przycisk "z menu czy coś takiego? "Dla nas teraz odpowiedź brzmi" nie ". Tak, ostatecznie nauczymy się używać kodu z biblioteki (i możesz użyć biblioteki do łatwiejszego tworzenia przycisków w szkicach), ale jest wiele wartości w nauce programowania GUI (graficzny interfejs użytkownika) elementy od zera. Po pierwsze, ćwiczenia przycisków programistycznych, najazdów i suwaków to doskonały sposób na poznanie podstaw zmiennych i warunków. A dwie, używając tych samych starych przycisków i najazdów, które każdy program ma, nie jest strasznie ekscytujący. Jeśli zależy ci na rozwijaniu nowych interfejsów, zrozumienie, jak zbudować interfejs od zera, to umiejętność, której będziesz potrzebować. OK, przy okazji, przyjrzymy się, w jaki sposób używamy zmiennej binarnej do zaprogramowania przycisku. Zmienna boolowska (lub zmienna typu boolean) jest zmienną, która może być tylko prawdą lub fałszem. Atrament to jako przełącznik. Jest włączony lub wyłączony. Naciśnij przycisk, włącz przełącznik. Naciśnij ponownie przycisk, wyłącz go. Po prostu użyliśmy zmiennej boolean w przykładzie 5-2: wbudowana zmienna mousePressed. mousePressed ma wartość true, gdy mysz jest naciśnięta, a false, gdy mysz nie jest. I tak nasz przykład przycisku będzie zawierał jedną zmienną binarną z początkową wartością false (przy założeniu, że przycisk zaczyna się w stanie wyłączonym).

```
boolean button = false; // zmienna boolean jest albo true albo false
```

W przypadku rolowania za każdym razem, gdy mysz unosiła się nad prostokątem, zmieniła kolor na biały. Nasz szkic zmieni tło na białe, gdy przycisk zostanie naciśnięty, a czarny, gdy nie jest.

```
if (button) {

background(255);

} else {  // jeśli wartość przycisku jest prawdziwa, tło jest białe. Jeśli jest fałszywy, czarny.

background(0);

}
```

Następnie możemy sprawdzić, czy położenie myszy znajduje się wewnątrz prostokąta i czy naciśnięto mysz, ustawiając odpowiednio wartość przycisku na wartość true lub false. Oto pełny przykład

Przykład 5-4: Przytrzymaj przycisk

```
boolean button = false;

int x = 50;

int y = 50;

int w = 100;

int h = 75;

void setup() {

size(200,200);
```

```
}

void draw() {

if (mouseX > x & & mouseX < x + w & & mouseY > y & & mouseY < y + h & & mousePressed) {

button = true;

} else {          // Przycisk jest wciśnięty, jeśli (mouseX, mouseY) znajduje się wewnątrz prostokąta,
                  //a mousePressed ma wartość true.

button = false;

}

if (button) {

background(255);

stroke(0);

} else {

background(0);

stroke(255);

}

fill(175);

rect(x,y,w,h);

}
```

Ten przykład symuluje przycisk podłączony do światła, które jest włączone tylko po naciśnięciu przycisku. Jak tylko puścisz, światło zgaśnie. Chociaż może to być doskonale odpowiednia forma interakcji dla niektórych instancji, nie jest to tym, co naprawdę zamierzamy w tej sekcji. To, czego chcemy, to przycisk działający jak przełącznik; po naciśnięciu przełącznika (naciśnij przycisk), jeśli lampka jest wyłączona, włącza się. Jeśli jest włączony, gaśnie. Aby to działało poprawnie, musimy sprawdzić, czy mysz znajduje się wewnątrz prostokąta wewnątrz mousePressed (), a nie jak wyżej w draw (). Z definicji, gdy użytkownik kliknie myszą, kod wewnątrz mousePressed () jest wykonywany raz i tylko jeden raz. Po kliknięciu myszą chcemy, aby przełącznik włączał się lub wyłączał (raz i tylko raz). Musimy teraz napisać kod, który "przełącza" przełącznik, zmienia jego stan z na wyłączony lub wyłączony na włączony. Th jest kod wejdzie wewnątrz mousePressed (). Jeśli zmienna "button" jest równa true, powinniśmy ustawić ją na false. Jeśli jest fałszywa, powinniśmy ustawić ją na true.

```
if (button) {

button = false;

} else {          // Wyraźny sposób przełączania zmiennej binarnej. Jeśli wartość przycisku ma wartość
                  //true, ustaw go na false. W przeciwnym razie musi być fałszywa, więc ustaw ją na true.

button = true;

}
```

Istnieje prostszy sposób, który jest następujący:

button =! button; // Nie prawda jest fałszem. Nie fałsz jest prawdą!

Tutaj wartość przycisku jest ustawiona na "nie". Innymi słowy, jeśli przycisk jest prawdziwy, ustawiamy go na nie true (false). Jeśli jest false, ustawiamy go na false (true). Uzbrojeni w tę dziwną, ale skuteczną linię kodu, jesteśmy gotowi spojrzeć na przycisk w akcji w Przykładzie 5-5.

Przykład 5-5: Przycisk jako przełącznik

```
boolean button = false;

int x = 50;

int y = 50;

int w = 100;

int h = 75;

void setup() {

size(200,200);

}

void draw() {

if (button) {

background(255);

stroke(0);

} else {

background(0);

stroke(255);

}

fill(175);            // Po naciśnięciu myszy następuje przełączenie stanu przycisku. Spróbuj przenieść ten
                      //kod do rysowania () jak w przykładzie najazdu

rect(x,y,w,h);

}

void mousePressed() {

if (mouseX > x & & mouseX < x + w & & mouseY > y & & mouseY < y + h){

button = !button;

}

}
```

Ćwiczenie 5-7: Dlaczego poniższy kod nie działa poprawnie, gdy jest przenoszony do funkcji draw ()?

```
if (mouseX > x && mouseX < x+w && mouseY > y && mouseY < y+h &&

mousePressed){

button = !button;

}
```

Ćwiczenie 5-8: Przykład 4-3 w poprzednim rozdziale przesunął koło przez okno. Zmień szkic tak, aby kółko zaczęło się poruszać dopiero po naciśnięciu myszy. Użyj zmiennej boolean.

```
boolean _____ = _____;

int circleX = 0;

int circleY = 100;

void setup() {

size(200,200);

}

void draw() {

background(100);

stroke(255);

fill(0);

ellipse(circleX,circleY,50,50);

_____

_____

_____

}

void mousePressed() {

_____

}
```

5.7 Odbijająca piłka

Nadszedł czas, aby powrócić do naszego przyjaciela Stworka. Zobaczmy, co zrobiliśmy do tej pory. Najpierw nauczyliśmy się rysować Stworka z funkcjami kształtów dostępnymi w referencji Przetwarzanie. Potem zdaliśmy sobie sprawę, że my może używać zmiennych zamiast wartości zakodowanych na stałe. Posiadanie tych zmiennych pozwoliło nam przenieść Zoog. Jeśli lokalizacja Zoog jest X, narysuj ją na X, następnie na X + 1, następnie na X+2 i tak dalej. To był ekscytujący, ale smutny moment. Ta przyjemność, którą odczuliśmy po odkryciu ruchu, szybko została zastąpiona przez samotne uczucie patrzenia Zooga na ekran. Na szczęście zapisy warunkowe są tutaj, aby uratować dzień, pozwalając nam zadać pytanie: Czy Zoog dotarł do krawędzi ekranu? Jeśli tak, odwróć Zooga! Aby uprościć wszystko, zacznijmy od prostego koła zamiast całego wzoru Zooga. Napisz program, w którym Zoog (prosty okrąg) porusza się po ekranie w poziomie od lewej do prawej strony. Kiedy

osiągnie prawą krawędź, odwraca kierunek. Z poprzedniego rozdziału dotyczącego zmiennych wiemy, że potrzebujemy zmiennych globalnych, aby śledzić lokalizację Stworka.

int x = 0;

Czy to wystarczy? Nie. W naszym poprzednim przykładzie Stworka zawsze przesuwał o jeden piksel.

x = x + 1;

To mówi Stworkowi, aby przesunął się w prawo. Ale co, jeśli chcemy, aby przesunął się w lewo? Łatwo, prawda?

x = x - 1;

Innymi słowy, czasami Stworek porusza się z prędkością od 1 do 1, a niekiedy 1. Prędkość Stworka zmienia się. Tak, dzwoni dzwonek. Aby zmienić kierunek prędkości Stworka, potrzebujemy kolejnej zmiennej: speed

int x = 0; // Zmienna speed dla Stworka. Gdy prędkość jest dodatnia Stworek przesuwa się w prawo, gdy prędkość jest ujemna Stworek przesuwa się w lewo.

int prędkość = l;

Teraz, gdy mamy nasze zmienne, możemy przejść do reszty kodu. Zakładając, że setup () ustawia rozmiar okna, możemy przejść bezpośrednio do sprawdzenia kroków wymaganych wewnątrz funkcji draw (). Możemy również odnieść się do Zooga jako kuli w tym przypadku, ponieważ właśnie narysujemy okrąg.

background(0);

stroke(255);

fill(100);

ellipse(x,100,32,32); // Dla uproszczenia Stworek to tylko koło.

Podstawowe rzeczy. Teraz, aby kulka się poruszyła, wartość jej x lokalizacji powinna się zmieniać w każdym cyklu przez losowanie ().

x = x + prędkość;

Gdybyśmy teraz uruchomili program, krąg zaczynałby się po lewej stronie okna, przesuwałby się w prawo i dalej poza krawędź ekranu - to jest rezultat, który osiągnęliśmy w Rozdziale 4. Aby się odwrócił , potrzebujemy warunkowego oświadczenia.

Jeśli piłka zejdzie z krawędzi, obróć piłkę.

Lub bardziej formalnie. . .

Jeśli x jest większe niż szerokość, prędkość wsteczna.

if (x > width) {

speed = speed * -1; // Mnożenie przez -1 odwraca prędkość.

}

Odwracanie biegunowości liczby

Kiedy chcemy odwrócić polaryzację liczby, mamy na myśli, że chcemy, by liczba dodatnia stała się ujemna, a liczba ujemna by stała się dodatnia. Osiąga się to przez pomnożenie przez -1.

Przypomnij sobie następujące rzeczy:

- -5 * -1 = 5
- -5 * -1 = -5
- -1 * 1 = -1
- -1 * -1 = 1

Uruchamiając szkic, mamy teraz okrąg, który obraca się, gdy osiągnie skrajną prawą krawędź, ale biegnie poza skrajną lewą krawędzią ekranu. Będziemy musieli nieco zmienić warunki. Jeśli piłka zejdzie z prawej lub lewej krawędzi, obróć piłkę dookoła. Lub bardziej formalnie. . .

Jeśli x jest większe niż szerokość lub jeśli x jest mniejsze od zera, prędkość wsteczna.

```
if ((x > width) || (x < 0)) { // Pamiętaj, || oznacza „lub”
speed = speed * -1;
}
```

Przykład 5-6 umieszcza to wszystko razem.

Przykład 5-6: Odbijająca się piłka

```
int x = 0;
int speed = 1;
void setup() {
size(200,200);
smooth();
}
void draw() {
background(255);
x = x + speed;   // Dodaj bieżącą prędkość do lokalizacji x.
if ((x > width) || (x < 0)) {
speed = speed * -1;  // Jeśli obiekt osiągnie jedną z krawędzi, pomnóż prędkość przez -1, aby go obrócić.
}
// Display circle at x location
stroke(0);
fill(175);
```

```
ellipse(x,100,32,32);

}
```

Ćwiczenie 5-9: PrzepiszPrzykład 5-6, aby piłka poruszała się nie tylko w poziomie, ale również w pionie. Czy możesz wdrożyć dodatkowe funkcje, takie jak zmiana rozmiaru lub koloru piłki na podstawie określonych warunków? Czy możesz przyspieszyć lub zwolnić piłkę oprócz zmiany kierunku?

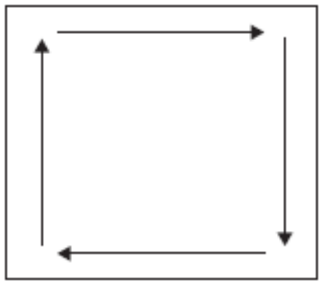
Logika "podskakującej kuli" inkrementacji i dekrementacji zmiennej może być stosowana na wiele sposobów poza ruch kształtów na ekranie. Na przykład, tak jak kwadrat przesuwa się od lewej do prawej, kolor może przejść od mniej czerwonego do bardziej czerwonego. Przykład 5-7 przyjmuje ten sam algorytm podskakującej piłki i stosuje go do zmiany koloru.

Przykład 5-7: Kolor "podskakujący"

```
float c1 = 0;

float c2 = 255; // Dwie zmienne dla koloru

float c1dir = 0.1;

float c2dir = -0.1; // Zacznij od zwiększania c1. Zacznij od zmniejszania c2.

void setup() {

size(200,200);

}

void draw() {

noStroke();

// Draw rectangle on left

fill(c1,0,c2);

rect(0,0,100,200);

// Draw rectangle on right

fill(c2,0,c1);

rect(100,0,100,200);

// Adjust color values

c1 = c1 + c1dir;

c2 = c2 + c2dir;

// Reverse direction of color change

if (c1 < 0 || c1 > 255) {

c1dir * = -1;      // Zamiast docierać do krawędzi okna, zmienne te osiągają "krawędź" koloru: 0 dla
                   //braku koloru i 255 dla pełnego koloru. Kiedy tak się dzieje, podobnie jak w przypadku
                   //odbijającej się piłki, kierunek jest odwrócony.

}
```

```
if (c2 < 0 || c2 > 255) {

c2dir * = -1;

}

}
```

Posiadanie instrukcji warunkowej w naszej kolekcji narzędzi programistycznych pozwala na bardziej złożony ruch. Rozważmy na przykład prostokąt następujący po krawędziach okna. Jednym ze sposobów rozwiązania tego problemu jest myślenie o ruchu prostokąta jako o czterech możliwych stanach, ponumerowanych od 0 do 3. Patrz rysunek



• Stan # 0: od lewej do prawej.

• Stan # 1: od góry do dołu.

• Stan # 2: od prawej do lewej.

• Stan # 3: od dołu do góry.

Możemy użyć zmiennej, aby śledzić numer stanu i dostosować współrzędne x, y prostokąta według stanu. Na przykład:

"Jeśli stan to 2, x oznacza x minus 1."

Gdy prostokąt dotrze do punktu końcowego dla tego stanu, możemy zmienić zmienną stanu. "Jeśli stan to 2:

(a) x jest równe x minus 1. (b) jeśli x jest mniejsze od zera, stan wynosi 3. "

Poniższy przykład implementuje tę logikę

Przykład 5-8: Kwadrat za krawędzią, używa zmiennej "stan"

```
int x = 0; // x location of square

int y = 0; // y location of square

int speed = 5; // speed of square

int state = 0;     // Zmienna do śledzenia stanu "kwadratu" kwadratu. W zależności od wartości jego
                   //stanu będzie albo przesuwać się w prawo, w dół, w lewo, albo w górę.

void setup() {

size(200,200);

}
```

```
void draw() {
background(100);
// Display the square
noStroke();
fill(255);
rect(x,y,10,10);
if (state = = 0) { // Jeśli stan wynosi 0, przejdź w prawo.
x = x + speed;
if (x > width-10) {        // Jeśli podczas gdy stan wynosi 0, osiąga prawą stronę okna, zmień stan na 1.
                           //Powtórz tę samą logikę dla wszystkich stanów!
x = width-10;
state = 1;
}
} else if (state = = 1) {
y = y + speed;
if (y > height-10) {
y = height-10;
state = 2;
}
} else if (state = = 2) {
x = x - speed;
if (x < 0) {
x = 0;
state = 3;
}
} else if (state = = 3) {
y = y - speed;
if (y < 0) {
y = 0;
state = 0;
}
```

}

}

5.8 Fizyka 101

Dla mnie jednym z najszczęśliwszych momentów mojego życia programistycznego była chwila, gdy zdałem sobie sprawę, że mogę zakodować grawitację. W rzeczywistości, uzbrojony w zmienne i warunkowe, jesteś teraz gotowy na tę chwilę. Skaczący szkic kuli uczył nas, że obiekt porusza się, zmieniając jego położenie zgodnie z prędkością.

location = location + speed

Grawitacja jest siłą przyciągania pomiędzy wszystkimi masami. Kiedy upuszczasz pióro, siła grawitacji z ziemi (która jest przytłaczająco większa niż długopis) powoduje, że pióro przyspiesza w kierunku ziemi. To, co musimy dodać do naszej odbijającej się piłki, to pojęcie "przyspieszenia" (które jest spowodowane grawitacją, ale może być spowodowane przez dowolną liczbę sił). Przyspieszenie zwiększa (lub zmniejsza) prędkość. Innymi słowy, przyspieszeniem jest szybkość zmiany prędkości. Szybkość to szybkość zmiany lokalizacji. Potrzebujemy tylko kolejnej linii kodu:

prędkość = prędkość + przyspieszenie

A teraz mamy prostą symulację grawitacji.

Przykład 5-9: Prosta grawitacja

```
float x = 100; // x location of square

float y = 0; // y location of square

float speed = 0; // speed of square

float gravity = 0.1;     // Nowa zmienna dla grawitacji (tzn. przyspieszenia). Używamy stosunkowo
                         //małej liczby (0,1), ponieważ przyspieszenie to narasta z biegiem czasu,
                         //zwiększając prędkość. Spróbuj zmienić tę liczbę na 2.0 i zobacz, co się stanie.

void setup() {

size(200,200);

}

void draw() {

background(255);

// Display the square

fill(0);

noStroke();

rectMode(CENTER);

rect(x,y,10 , 10);

y = y + speed; // Dodaj prędkość do lokalizacji

speed = speed + gravity;  // Dodaj grawitację do prędkości.
```

```
// If square reaches the bottom

// Reverse speed

if (y > height) {

speed = speed * -0.95;  //Mnożenie przez -0.95 zamiast -1 spowalnia kwadrat za każdym razem, gdy
                        //się odskakuje (przez zmniejszenie prędkości). Jest to znane jako efekt
                        //"tłumienia" i jest bardziej realistyczną symulacją prawdziwego świata (bez
                        //niego piłka podskoczyłaby na zawsze).

}

}
```

Ćwiczenie 5-10: Kontynuuj projekt i dodaj niektóre funkcje przedstawione w tej części. Niektóre opcje:

• Wykonuj części swoich najazdów projektowych, które zmieniają kolor, gdy mysz znajduje się nad wybranymi obszarami.

• Przesuń go po ekranie. Czy możesz sprawić, by odbijał się od wszystkich krawędzi okna?

• Zmieniaj i zmniejszaj kolory.

Oto prosta wersja ze Stworkiem.

Przykład 5-10: Stworek i warunki

```
loat x = 100;

float y = 100;

float w = 60;

float h = 60;

float eyeSize = 16;

float xspeed = 3;

float yspeed = 1; // Stworek ma zmienne prędkości w kierunku poziomym i pionowym.

void setup() {

size(200,200);

smooth();

}

void draw() {

// Change the location of Zoog by speed

x = x + xspeed;

y = y + yspeed;
```

```
if ((x > width) || (x < 0)) {          //Instrukcja IF z logicznym OR określa, czy Stworka osiągnął prawą lub
                                       //lewą krawędź ekranu. Gdy to prawda, mnożymy prędkość przez 1,
                                       //odwracając kierunek Stworka!
xspeed = xspeed * -1;
}
if ((y > height) || (y < 0)) {         // Identyczna logika jest również stosowana w kierunku y.
yspeed = yspeed * -1;
}
background(0);
ellipseMode(CENTER);
rectMode(CENTER);
noStroke();
// Draw Zoog's body
fill(150);
rect(x,y,w/6,h*2);
// Draw Zoog's head
fill(255);
ellipse(x,y-h/2,w,h);
// Draw Zoog's eyes
fill(0);
ellipse(x-w/3,y-h/2,eyeSize,eyeSize*2);
ellipse(x + w/3,y-h/2,eyeSize,eyeSize*2);
// Draw Zoog's legs
stroke(150);
line(x-w/12,y + h,x-w/4,y + h + 10);
line(x + w/12,y + h,x + w/4,y + h + 10);
}
```

VI Pętle

6.1 Co to jest iteracja? Mam na myśli, co to jest iteracja? Poważnie, co to jest iteracja?

Iteracja to generatywny proces wielokrotnego powtarzania zestawu reguł lub kroków. Jest to fundamentalna koncepcja w programowaniu komputerowym i wkrótce odkryjemy, że czyni to nasze życie jako programistów całkiem uroczymi. Zaczynajmy. W tej chwili pomyśl o nogach. Dużo i dużo nóg na naszym małym Stworku. Gdybyśmy przeczytali tylko część 1 tej książki, prawdopodobnie napisalibyśmy kod jak w przykładzie 6-1

Przykład 6-1: Wiele linii

```
size(200,200);

background(255);

// Legs

stroke(0);

line(50,60,50,80);

line(60,60,60,80);

line(70,60,70,80);

line(80,60,80,80);

line(90,60,90,80);

line(100,60,100,80);

line(110,60,110,80);

line(120,60,120,80);

line(130,60,130,80);

line(140,60,140,80);

line(150,60,150,80);
```

W powyższym przykładzie nogi są rysowane od x = 50 pikseli do x = 150 pikseli, z jedną nogą co 10 pikseli. Oczywiście, kod osiąga to, jednakże, mając wyuczone zmienne w Części 4, możemy dokonać pewnych istotnych ulepszeń i wyeliminować zakodowane wartości. Najpierw konfigurujemy zmienne dla każdego parametru naszego systemu: położenie nóg x, y, długość i odległość między nogami. Zauważ, że dla każdej narysowanej nogi zmienia się tylko wartość x. Wszystkie pozostałe zmienne pozostają takie same (ale mogą się zmienić, jeśli chcemy je!).

Przykład 6-2: Wiele linii ze zmiennymi

```
size(200,200);

background(0);

// Legs

stroke(255);
```

```
int y = 80; // Pionowe położenie każdej linii

int x = 50; // Początkowa lokalizacja pozioma dla pierwszej linii

int spacing = 10; // Jak daleko od siebie jest każda linia

int len = 20; // Długość każdej linii

line(x,y,x,y + len); // Narysuj pierwszą nogę.

x = x + spacing; // Dodaj odstępy, aby następna noga pojawiła się 10 pikseli po prawej stronie

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len); // Kontynuuj ten proces dla każdej nogi, powtarzając ją w kółko.

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);

x = x + spacing;

line(x,y,x,y + len);
```

Nie jest tak źle, jak sądzę. O dziwo, chociaż jest to technicznie bardziej efektywne (możemy zmienić zmienną odległości, na przykład zmieniając tylko jedną linię kodu), zrobiliśmy krok wstecz, tworząc dwukrotnie więcej kodu! A co jeśli chcemy narysować 100 nóg? Do każdej nogi potrzebujemy dwóch linii kodu. Th at ma 200 linii kodu na 100 nóg! Aby uniknąć tego tragicznego problemu wywołującego tunel nadgarstka, chcemy być w stanie powiedzieć coś takiego:

Narysuj jedną linię sto razy.

Aha, tylko jedna linia kodu! Oczywiście, nie jesteśmy pierwszymi programistami, którzy sięgnęliby po ten dylemat i łatwo można go rozwiązać za pomocą bardzo powszechnie stosowanej struktury

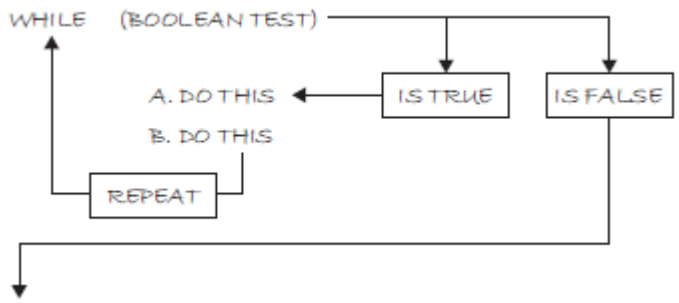
kontrolnej - pętli. Struktura pętli jest podobna w składni do warunkowej. Nie jest tak źle, jak sądzę. O dziwo, chociaż jest to technicznie bardziej efektywne (możemy zmienić zmienną odległości, na przykład zmieniając tylko jedną linię kodu), zrobiliśmy krok wstecz, tworząc dwukrotnie więcej kodu! A co jeśli chcemy narysować 100 nóg? Do każdej nogi potrzebujemy dwóch linii kodu. Th at ma 200 linii kodu na 100 nóg! Aby uniknąć tego tragicznego problemu wywołującego tunel nadgarstka, chcemy być w stanie powiedzieć coś takiego:

Narysuj jedną linię sto razy.

Aha, tylko jedna linia kodu! Oczywiście, nie jesteśmy pierwszymi programistami, którzy sięgnęliby po ten dylemat i łatwo można go rozwiązać za pomocą bardzo powszechnie stosowanej struktury kontrolnej - pętli. Struktura pętli jest podobna w składni do warunkowej. Jednak zamiast zadawać pytania typu "tak" lub "nie", aby ustalić, czy blok kodu powinien zostać wykonany jeden raz, nasz kod zapyta "tak" lub "nie", aby określić, ile razy blok kodu powinien zostać powtórzony. Th jest znany jako iteracja.

6.2 Pętla "WHILE", jedyna pętla, której naprawdę potrzebujesz

Są trzy rodzaje pętli, pętla while, pętla do-while i pętla for. Na początek skupimy się na chwilę na pętli while (przepraszam, nie mogłem się oprzeć). Z jednej strony, jedyna pętla naprawdę potrzebna jest chwila. Ta pętla, jak zobaczymy, jest po prostu wygodną alternatywą, świetnym skrótem do prostych operacji liczenia. Do-while jednak jest rzadko używany (nie jest to jeden z przykładów w tej książce), więc go zignorujemy. Podobnie jak w przypadku struktur warunkowych (jeśli / poza), pętla while stosuje boolowskie warunki testowe. Jeśli test zostanie oceniony na true, instrukcje zawarte w nawiasach klamrowych są wykonywane; jeśli jest fałszywa, przechodzimy do następnego wiersza kodu. Różnica polega na tym, że instrukcje wewnątrz bloku while są nadal wykonywane wielokrotnie, dopóki warunek testowy nie stanie się fałszywy. Patrz rysunek



Weźmy kod z problemu nóg. Zakładając następujące zmienne. . .

int y = 80; // Pionowe położenie każdej linii

int x = 50; // Początkowa lokalizacja pozioma dla pierwszej linii

int spacing = 10; // Jak daleko od siebie jest każda linia

int len = 20; // Długość każdej linii

... musieliśmy ręcznie powtórzyć następujący kod:

stroke (255);

line (x, y, x, y + len); // Narysuj pierwszą nogę

x = x + spacing; // Dodaj "odstęp" do x

line (x, y, x, y + len); // Kolejna noga ma 10 pikseli po prawej

x = x + spacing; // Dodaj "odstęp" do x

line (x, y, x, y + len); // Kolejna noga ma 10 pikseli po prawej

x = x + spacing; // Dodaj "spacing" do x

line(x, y, x, y + len); // Kolejna noga ma 10 pikseli po prawej

// itd. itp. powtarzanie z nowymi nogami

Teraz, mając wiedzę o istnieniu pętli while, możemy przepisać kod tak, jak w przykładzie 6-3, dodając zmienną, która mówi nam, kiedy przerwać zapętlanie, czyli na jakim pikselu kończy się noga.

Przykład 6-3: Pętla while

int endLegs = 150; // Zmienna do oznaczania końca nóg.

stroke(0);

while (x < = endLegs) {

line (x,y,x,y + len); // Narysuj każdą nogę wewnątrz pętli.

x = x + spacing;

}

Zamiast pisać "line (x, y, x, y + len); "Wiele razy, jak to zrobiliśmy w pierwszej kolejności, teraz piszemy to tylko raz w pętli while, mówiąc" dopóki x ma mniej niż 150, narysuj linię na x, cały czas zwiększając x. "I tak, co wcześniej zajęło 21 linii kodu, teraz zajmuje tylko cztery! Ponadto możemy zmienić zmienną odległości, aby wygenerować więcej nóg. Wyniki pokazano na rysunku

int spacing = 4; // Mniejsza wartość odstępu powoduje, że nogi są bliżej siebie.

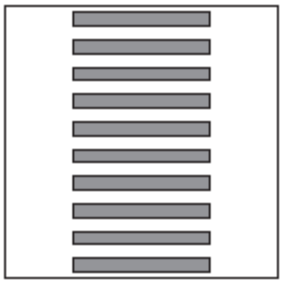
while (x < = endLegs) {

line (x,y,x,y + len); // Draw EACH leg

x = x + spacing;

}

Spójrzmy na jeszcze jeden przykład, tym razem używając prostokąta zamiast linii, jak pokazano na rysunku poniżej, i zadaj trzy kluczowe pytania



```
while (y < 100) {

// Loop!

}
```

3. Jaka jest twoja operacja pętli? W tym przypadku za każdym razem, gdy będziemy w pętli, chcemy narysować nowy prostokąt poniżej poprzedniego. Możemy to osiągnąć, wywołując funkcję rect () i zwiększając y o 20.

```
rect(100,y,100,10);

y = y + 20;
```

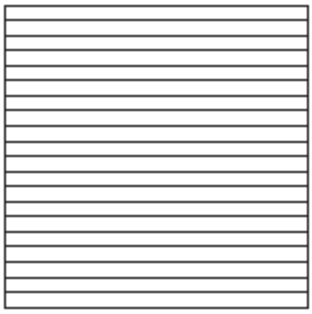
Wstawiając wszystko razem:

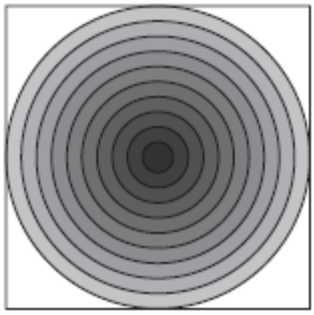
```
int y = 10;   // Stan początkowy.

while (y < height) {      // Pętla jest kontynuowana, gdy wyrażenie logiczne jest prawdziwe. Dlatego
                          //pętla zatrzymuje się, gdy wyrażenie logiczne ma wartość false.

rect(100,y,100,10);

y = y + 20;       // Każdorazowo zwiększamy y za pomocą pętli, rysując prostokąt po prostokącie, aż y
                  nie będzie już mniejsze niż wysokość.

}
```

Ćwiczenie 6-1: Wypełnij puste miejsca w kodzie, aby odtworzyć następujące zrzuty ekranu.

```
size(200,200);

background(255);

int y = 0;

while (_________) {

stroke(0);

line(_________,_________,_________,_________);

y = _________ ;

}

size(200,200);

background(255);
```

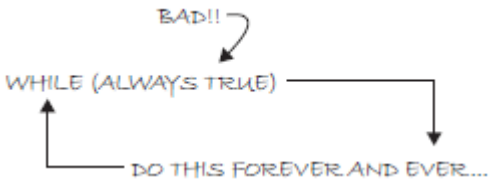
```
float w = _________ ;

while (_________) {

stroke(0);

fill(_________);

ellipse(_________,_________,_________,_________);

__________20;

}
```





6.3 Warunki "wyjścia"

Pętle, jak zapewne zaczynacie rozumieć, są bardzo przydatne. Niemniej jednak w świecie pętli jest ciemne, podejrzane podbrzusze, gdzie żyją paskudne rzeczy znane jako nieskończone pętle. Patrz rysunek



Badając "nogi" w przykładzie 6-3, widzimy, że gdy tylko x jest większe niż 150, pętla zatrzymuje się. I zawsze tak się dzieje, ponieważ x zwiększa się o "spacing", który jest zawsze liczbą dodatnią. To nie jest wypadek; za każdym razem, gdy uruchamiamy programowanie z pętlą, musimy upewnić się, że warunek wyjścia dla pętli zostanie ostatecznie spełniony!

Przetwarzanie nie spowoduje błędu, jeśli warunki wyjścia nigdy nie wystąpią. Rezultatem jest Sisyphean, ponieważ twoja pętla rzuca głazem w górę w kółko w nieskończoność.

Przykład 6-4: Pętla nieskończona. Nie rób tego!

```
int x = 0;

while (x <10) {

println (x);

x = x - 1;          // Zmniejszanie x daje w tym miejscu pętlę infinitową, ponieważ wartość x nigdy nie
                    //będzie większa niż 10. Bądź ostrożny!

}
```

W przypadku kopnięć spróbuj uruchomić powyższy kod (upewnij się, że zapisałeś całą swoją pracę i nie uruchamiasz żadnego innego krytycznego oprogramowania na twoim komputerze). Szybko zobaczysz, że przetwarzanie się zawiesza. Jedynym wyjściem z tej sytuacji jest prawdopodobnie przerwanie procesu. Inflite loopy nie są często tak oczywiste, jak w przykładzie 6-4. Oto kolejny błędny program, który czasami powoduje awarię pętli nieskończonej.

Przykład 6-5: Kolejna pętla nieskończona Nie rób tego!

```
int y = 80; // Pionowe położenie każdej linii

int x = 0; // Pozioma lokalizacja pierwszego wiersza

int spacing = 10; // Jak daleko od siebie jest każda lini

int len = 20; // Długość każdej linii

int endLegs = 150; // Gdzie powinny się zatrzymać linie?

void setup() {

size(200,200);

}

void draw() {

background(0);

stroke(255);

x = 0;

spacing = mouseX / 2;   // Zmiennej odległości, która określa odległość pomiędzy liniami, przypisuje się
                        //wartość równą mouseX podzieloną przez dwa.

while (x < = endLegs) { // Warunek wyjścia - gdy x jest większe niż końcówki

line(x,y,x,y + len);

x = x + spacing;  // Zwiększenie x. x zawsze zwiększa się o wartość odstępów. Jaki jest zakres możliwych
                  //wartości dla odstępów?

}

}
```

Czy pojawi się nieskończona pętla? Wiemy, że utkniemy w pętli na zawsze, jeśli x nigdy nie jest większe niż 150. A ponieważ x zwiększa się o odstępy, jeśli odstęp wynosi zero (lub liczba ujemna) x zawsze pozostanie taka sama wartość (lub zejdzie na wartość). Przywoływanie Funkcja constrain() opisana w części 4, możemy zagwarantować brak nieskończonej pętli przez ograniczenie wartości odstępów do dodatniego zakresu liczb:

```
int spacing = constrain (mouseX / 2, 1, 100);     // Używając metody constrain (), aby upewnić się, że
                                                   //warunek wyjścia jest spełniony.
```

Ponieważ odstępy są bezpośrednio związane z koniecznymi warunkami wyjścia, wymuszamy określony zakres wartości, aby upewnić się, że nie ma nigdy nieskończonej pętli. Innymi słowy, w pseudokodie mówimy: "Narysuj serię linii rozstawionych przez N pikseli, gdzie N nigdy nie może być mniejsze niż 1! "Jest to również użyteczny przykład, ponieważ ujawnia interesujący fakt o mouseX. Możesz ulec pokusie, aby spróbować umieścić mouseX bezpośrednio w wyrażeniu inkrementacji w następujący sposób:

```
while (x < = endLegs) {

line(x,y,x,y + len);  // Umieszczenie myszy w pętli nie jest rozwiązaniem problemu z nieskończoną pętlą

x = x + mouseX / 2;

}
```

Czy to nie rozwiązałoby problemu, skoro nawet pętla utknie, gdy tylko użytkownik przeniesie mysz do poziomej lokalizacji większej od zera, warunek wyjścia zostanie spełniony? To miła myśl, ale ta, która jest bardzo niefortunna. mouseX i mouseY są aktualizowane o nowe wartości na początku każdego cyklu poprzez draw (). Zatem nawet jeśli użytkownik przesunie mysz do lokalizacji X50 z lokalizacji 0, mouseX nigdy nie pozna tej nowej wartości, ponieważ utknie ona w jej pętli infinit i nie będzie w stanie przejść do następnego cyklu przez draw().

6.4 Pętla "FOR"

Pewien styl pętli while, w którym jedna wartość jest wielokrotnie zwiększana, jest szczególnie powszechny. Będzie to jeszcze bardziej widoczne, gdy spojrzymy na tablice w części 9. Pętla for jest skrótem do powszechnie występujących pętli. Zanim przejdziemy do szczegółów, porozmawiajmy przez niektóre wspólne pętle, które możesz napisać w Processing i jak są napisane jako pętlę for.

Zacznij od 0 i policz do 9. for (int i = 0; i <10; i = i + 1)

Zacznij od 0 i zliczaj do 100 przez 10. for (int i = 0; i <101; i = i + 10)

Zacznij od 100 i odliczaj do 0 przez 5. for (int i = 100; i> = 0; i = i - 5)

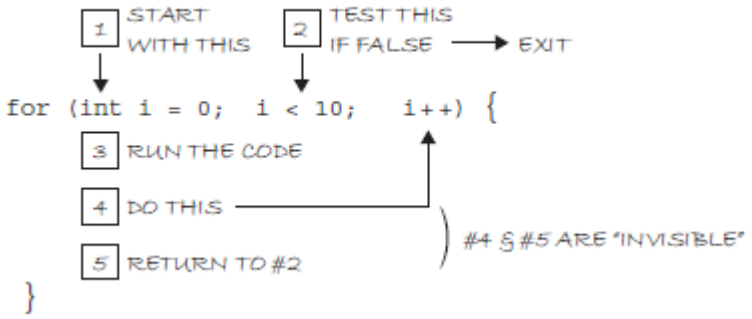
Patrząc na powyższe przykłady, widzimy, że pętla for składa się z trzech części:

• Inicjalizacja - tutaj deklarowana jest zmienna i inicjalizowana do użycia w ciele pętli. Ta zmienna jest najczęściej używana w pętli jako licznik.

• Test Boolean - Jest to dokładnie to samo, co testy boolowskie znalezione w instrukcjach warunkowych i pętlach while. Może to być dowolne wyrażenie, które zwraca wartość true lub false.

• Wyrażenie Iteracji - Ostatnim elementem jest instrukcja, która ma się zdarzyć z każdym cyklem pętli. Zauważ, że instrukcja jest wykonywana na końcu każdego cyklu przez pętlę. (Możesz mieć wielokrotne

wyrażenia iteracyjne, jak również zmienne inicjalizacje, ale dla uproszczenia nie będziemy się teraz o to martwić.)



W języku angielskim powyższy kod oznacza: powtórz ten kod 10 razy. Lub, mówiąc jeszcze prostiej: liczymy od zera do dziewięciu!

Do maszyny oznacza to:

• Zadeklaruj zmienną i i ustaw jej początkową wartość na 0.

• Gdy ja jest mniejsze niż 10, powtórz ten kod.

• Na końcu każdej iteracji dodaj jeden do i.

Pętla for może mieć własną zmienną tylko w celu zliczania. Zmienna, która nie została zadeklarowana u góry kodu, nazywana jest zmienną lokalną. Wyjaśnimy i krótko ją zdefiniujemy

Operatory inkrementacji / dekrementacji

Skrót do dodawania lub odejmowania jednego ze zmiennych jest następujący:

x ++; jest równoważne: x = x+ 1; co oznacza: przyrost o 1 lub dodaj 1 do bieżącej wartości x

x -- 1 jest równoważne: x = x - 1;

Mamy też:

x += 2; tak samo jak x = x + 2;

x *= 3; tak samo jak x = x * 3;

i tak dalej.

Ta sama pętla może być zaprogramowana w formacie while:

int i = 0;

while (i < 10) { // To jest tłumaczenie pętli for, używając pętli while.

i + + ;

// linie kodu do wykonania tutaj

}

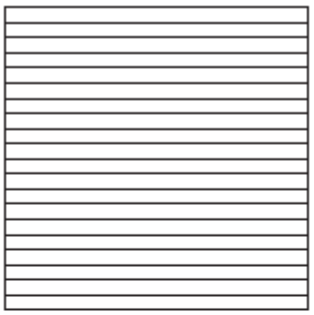
Przepisanie kodu rysunku nogi w celu użycia instrukcji for wygląda następująco:

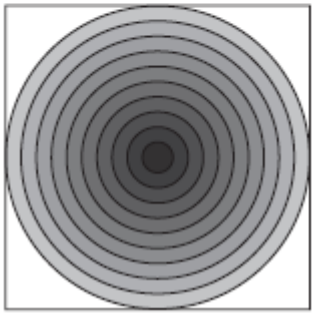
Przykład 6-6: Nogi z pętlą for

```
int y = 80; // Pionowe położenie każdej linii
int spacing = 10; // Jak daleko od siebie jest każda linia
int len = 20; // Długość każdej linii
for (int x = 50; x < = 150; x + = spacing) { // Tłumaczenie nóg z pętli while do pętli for.
line(x,y,x,y + len);
}
```

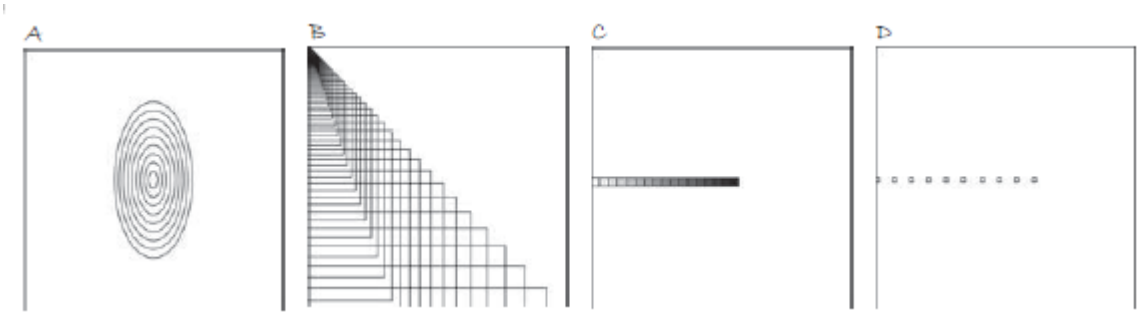
Ćwiczenie 6-2: Przepisz ćwiczenie 6-1, używając pętli for.

```
size(200,200);
background(255);
for (int y =_____;_____;_____) {
stroke(0);
line(_____,_____,_____,_____);
}
size(200,200);
background(255);
for (_____;_____;_____–  20) {
stroke(0);
fill(_____);
ellipse(_____,_____,_____,
_____);
ellipse(_____,_____,_____,
_____);
}
```





: Poniżej przedstawiono kilka dodatkowych przykładów pętli. Dopasuj odpowiedni zrzut ekranu ze strukturą pętli. Każdy przykład zakłada te same cztery linie początkowego kodu.



size(300,300); // Just setting up the size

background(255); // Black background

stroke(0); // Shapes have white lines

noFill(); // Shapes are not filled in

_____ for (int i = 0; i < 10; i++) {

rect(i*20,height/2, 5, 5);

}

_____ int i = 0;

while (i 10) {

ellipse(width/2,height/2, i*10, i*20);

i++;

```
}

_____ for (float i = 1.0; i < width; i *= 1.1) {

rect(0,i,i,i*2);

}

_____ int x = 0;

for (int c = 255; c > 0; c –= 15) {

fill(c);

rect(x,height/2,10,10);

x = x += 10;

}
```

6.5 Zmienne lokalne a globalne (zmienny zakres AKA)

Do tej chwili, za każdym razem, gdy użyliśmy zmiennej, zadeklarowaliśmy ją na początku naszego programu powyżej setup().

```
int x = 0;  // Zawsze deklarowaliśmy nasze zmienne u góry naszego kodu.

void setup () {

}
```

To było ładne uproszczenie i pozwoliło nam skupić się na podstawach deklarowania, inicjowania i używania zmiennych. Zmienne można jednak zadeklarować w dowolnym miejscu w ramach programu, a teraz przyjrzymy się, co to znaczy zadeklarować zmienną gdzieś indziej niż górna i jak możemy wybrać wybór właściwej lokalizacji dla deklarowania zmiennej. Wyobraźcie sobie przez chwilę, że program komputerowy kieruje waszym życiem. W tym życiu zmienne to fragmenty danych pisanych na post-it, które trzeba zapamiętać. Jeden post może mieć adres restauracji na lunch. Zapisujesz to rano i wyrzucasz po miłym burgeru z indyka. Ale inny post-it może zawierać kluczowe informacje (takie jak numer konta bankowego) i można go zachować w bezpiecznym miejscu przez wiele lat. Jest to pojęcie zakresu. Niektóre zmienne istnieją (tj. Są dostępne) przez cały czas trwania zmiennych globalnych programu - a niektóre z nich działają tymczasowo, tylko przez krótki moment, kiedy ich wartość jest wymagana dla instrukcji lub obliczeń - zmiennych lokalnych. W Processing zmienne globalne są zadeklarowane u góry programu, poza ustawieniami setup () i draw (). Te zmienne mogą być używane w dowolnym wierszu kodu w dowolnym miejscu w programie. Jest to najprostszy sposób użycia zmiennej, ponieważ nie musisz pamiętać, kiedy możesz i nie możesz używać tej zmiennej. Zawsze możesz użyć tej zmiennej (i dlatego zaczęliśmy od zmiennych globalnych). Zmienne lokalne to zmienne zadeklarowane w bloku kodu. Do tej pory widzieliśmy wiele różnych przykładów bloków kodu: setup (), draw (), mousePressed () i keyPressed (), instrukcje if oraz while i for loop. Zmienna lokalna zadeklarowana w bloku kodu jest dostępna tylko do użytku w tym specyficznym bloku kodu, w którym została zadeklarowana. Jeśli spróbujesz uzyskać dostęp do zmiennej lokalnej poza blokiem, w którym została zadeklarowana, otrzymasz następujący błąd:

"Nie znaleziono dostępnego pola o nazwie" variableName ""

Jest to dokładnie ten sam dokładny błąd, który wystąpiłby, gdybyś nie zadeklarował w ogóle zmiennej "variableName". Przetwarzanie nie wie, co to jest, ponieważ żadna zmienna o tej nazwie nie istnieje w

bloku kodu, w którym się znajdujesz. Oto przykład użycia zmiennej lokalnej wewnątrz funkcji draw () w celu wykonania pętli while.

Przykład 6-7: Zmienna lokalna

```
void setup() {  // X jest niedostępny! Jest lokalny dla bloku kodu draw ()

size(200,200);

}

void draw() {

background(0);

int x = 0;        // X jest dostępny! Ponieważ jest on zadeklarowany w ramach bloku kodu draw (), jest
                  //dostępny tutaj. Zauważ jednak, że nie jest dostępny wewnątrz funkcji draw ()
                  //powyżej, gdzie jest zadeklarowany. Ponadto jest dostępny wewnątrz bloku while,
                  //ponieważ znajduje się wewnątrz funkcji draw().

while (x < width) {

stroke(255);

line(x,0,x,height);

x + = 5;

}

}

void mousePressed() {  // X jest niedostępny! Jest lokalny dla bloku kodu raw ()

println( "The mouse was pressed! ");

}
```

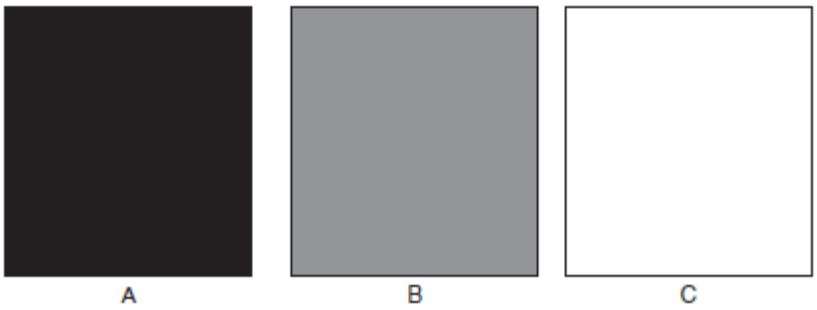
Po co się męczyć? Czy nie możemy po prostu zadeklarować x jako zmiennej globalnej? Chociaż jest to prawdą, ponieważ używamy tylko x wewnątrz funkcji draw (), marnowanie jej na zmienność globalną jest marnotrawstwem. Jest bardziej efektywny i ostatecznie mniej kłopotliwy przy programowaniu do deklarowania zmiennych tylko w zakresie, w którym są one konieczne. Z pewnością wiele zmiennych musi mieć charakter globalny, ale tak nie jest w tym przypadku. Pętla for wypala miejsce dla zmiennej lokalnej w części "inicjalizacja":

```
for (int i = 0; i < 100; i + = 10) {  // i jest dostępne tylko wewnątrz pętli for.

stroke(255);

fill(i);

rect(i,0,10,height);

}
```

Nie jest wymagane użycie zmiennej lokalnej w pętli for, jednak zwykle wygodnie jest to zrobić. Teoretycznie możliwe jest zadeklarowanie zmiennej lokalnej o tej samej nazwie, co zmienna globalna. W takim przypadku program użyje zmiennej lokalnej w ramach bieżącego zakresu i zmiennej globalnej

poza tym zakresem. Generalnie lepiej jest nigdy nie zadeklarować wielu zmiennych o tej samej nazwie, aby unikać tego rodzaju zamieszania.

Ćwiczenie 6-4: Przewidź wyniki następujących dwóch programów. Sprawdź swoją teorię, uruchamiając je.

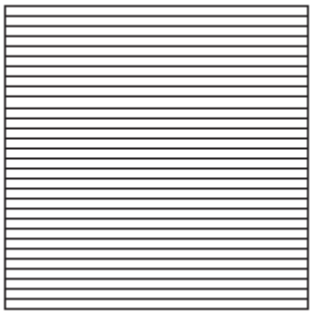



```
//SKETCH #1: Global
"count"
int count = 0;

void setup() {
  size(200,200);
}

void draw() {
  count = count + 1;
  background(count);
}
```

```
//SKETCH #2: Local
"count"
void setup() {
  size(200,200);
}

void draw() {
  int count = 0;
  count = count = 1;
  background(count);
}
```

6.6 Pętla wewnątrz głównej pętli

Rozróżnienie zmiennych lokalnych i globalnych przesuwa nas o krok dalej w kierunku pomyślnej integracji struktury pętli w Zoog. Zanim zakończymy ten rozdział, chciałbym rzucić okiem na jeden z najczęstszych punktów dezorientacji, który pojawia się przy pisaniu pierwszej pętli w kontekście "dynamicznego" szkicu przetwarzania. Rozważ następującą pętlę. Wynik pętli pokazano na rysunku



for (int y = 0; y < height; y + = 10) {

```
stroke (0);

line (0, y, width, y);

}
```

Powiedzmy, że chcemy wziąć powyższą pętlę i wyświetlić każdą linię po kolei, aby zobaczyć, że linie są animowane od góry do dołu. Naszą pierwszą myślą może być wykonanie powyższej pętli i przeniesienie jej do dynamicznego szkicu Processing za pomocą setup() i draw ()

```
void setup() {

size(200,200);

}

void draw() {

background(255);

for (int y = 0; y < height; y + = 10) {

stroke(0);

line(0,y,width,y);

}

}
```

Jeśli odczytamy kod, wydaje się sensowne, że zobaczymy każdą linię pojawiającą się pojedynczo. "Ustaw okno o wielkości 200 na 200 pikseli. Narysuj czarne tło. Narysuj linię na y równym 0. Narysuj linię na y równym 10. Narysuj linię na y równym 20. "Odnosząc się do Części 2, jednak pamiętamy, że Przetwarzanie faktycznie nie aktualizuje okna wyświetlacza do końca losowania ( ) został osiągnięty. Jest to niezbędne, aby pamiętać podczas korzystania z pętli i podczas pętli. Pętle te służą do powtarzania czegoś w kontekście jednego cyklu przez draw (). Są pętlą wewnątrz głównej pętli szkicu, draw (). Wyświetlanie linii jeden po drugim jest czymś, co możemy zrobić ze zmienną globalną w połączeniu z samą pętlą natury samego draw ()

Przykład 6-8: Linie pojedynczo

```
int y = 0 ;  // Tutaj nie ma pętli for. Zamiast tego zmienna globalna

void setup() {

size(200,200);

background(0);

frameRate(5); // Zwalnianie klatek na sekundę, abyśmy mogli łatwo zobaczyć efekt.

}

void draw() {

// Draw a line

stroke(255);
```

```
line(0,y,width,y);

// Increment y

y + = 10; // Za każdym razem, poprzez draw (), rysowana jest tylko jedna linia.

}
```

Logika tego szkicu jest identyczna jak w przykładzie 4 - 3, nasz pierwszy szkic ruchu ze zmiennymi. Zamiast przesuwać kółko po oknie w poziomie, przesuwajmy linię pionowo (ale nie usuwamy tła dla każdej ramki).

Ćwiczenie 6-5: Możliwe jest uzyskanie efektu renderowania jednej linii za pomocą pętli for. Sprawdź, czy możesz dowiedzieć się, jak to się robi. Część kodu znajduje się poniżej.

```
int endY;

void setup() {

size(200,200);

frameRate(5);

endY = _______;

}

void draw() {

background(0);

for (int y = _______; _________; _________) {

stroke(255);

line(0,y,width,y);

}

_______;

}
```

Korzystanie z pętli wewnątrz funkcji draw () również otwiera możliwość interaktywności. Przykład 6-9 wyświetla serię prostokątów (od lewej do prawej), z których każda jest zabarwiona jasnością w zależności od odległości od myszy

Przykład 6-9: Prosta pętla z interaktywnością

```
void setup() {

size(255,255);

background(0);

}

void draw() {

background(0);
```

```
// Start with i as 0

int i = 0;

// While i is less than the width of the window

while (i < width) {

noStroke();

float distance = abs(mouseX – i);        // Odległość między bieżącym prostokątem a myszą jest równa
                                         //bezwzględnej wartości różnicy między i i mouseX.

fill(distance);

rect(i,0,10,height);     //Ta odległość jest używana do wypełnienia koloru prostokąta w położeniu
                         //poziomym.

// Increase i by 10

i + = 10;

}

}
```

Ćwiczenie 6-6: Przepisz Przykład 6-9 za pomocą pętli for.

6.7 Stworek powiększa ramiona.

Ostatnim razem zostawiliśmy Stworka odbijającego się w naszym oknie Processing. To nowa wersja Stworka jedna mała zmiana. Przykład 6-10 używa pętli for do dodania serii linii do ciała Stworka, przypominających ramiona.

Przykład 6-10: Stworek z ramionami

```
int x = 100;

int y = 100;

int w = 60;

int h = 60;

int eyeSize = 16;

int speed = 1;

void setup() {

size(200,200);

smooth();

}

void draw() {

// Change the x location of Zoog by speed
```

```
x = x + speed;
// If we've reached an edge, reverse speed (i.e. multiply it by –1)
//(Note if speed is a + number, square moves to the right,– to the left)
if ((x    width)  (x < 0)) {
speed = speed * –1;
}
background(255); // Draw a white background
// Set ellipses and rects to CENTER mode
ellipseMode(CENTER);
rectMode(CENTER);
// Draw Zoog's arms with a for loop
for (int i = y + 5; i < y + h; i + = 10) {        // Ramiona są włączone do projektu Zooga za pomocą pętli
                                                  //for, która rysuje serię linii.
stroke(0);
line(x–w/3,i,x + w/3,i);
}
// Draw Zoog's body
stroke(0);
fill(175);
rect(x,y,w/6,h*2);
// Draw Zoog's head
fill(255);
ellipse(x,y–h/2,w,h);
// Draw Zoog's eyes
fill(0);
ellipse(x–w/3,y–h/2,eyeSize,eyeSize*2);
ellipse(x + w/3,y–h/2,eyeSize,eyeSize*2);
// Draw Zoog's legs
stroke(0);
line(x–w/12,y + h,x–w/4,y + h + 10);
line(x + w/12,y + h,x + w/4,y + h + 10);
```

```
}
```

Możemy również użyć pętli do narysowania wielu instancji Stworka umieszczając kod dla ciała Stworka wewnątrz pętli for

Przykład 6-11: Wiele Stworków

```
int w = 60;

int h = 60;

int eyeSize = 16;

void setup() {

size(400,200);

smooth();

}

void draw() {

background(255);

ellipseMode(CENTER);

rectMode(CENTER);

int y = height/2;

// Multiple versions of Zoog

for (int x = 80; x < width; x + = 80) {        // Zmienna x jest teraz zawarta w pętli for, w celu iteracji i
                                                //wyświetlania wielu Stworków

// Draw Zoog's body

stroke(0);

fill(175);

rect(x,y,w/6,h*2);

// Draw Zoog's head

fill(255);

ellipse(x,y–h/2,w,h);

// Draw Zoog's eyes

fill(0);

ellipse(x–w/3,y–h/2,eyeSize,eyeSize*2);

ellipse(x + w/3,y–h/2,eyeSize,eyeSize*2);

// Draw Zoog's legs

stroke(0);
```

line(x–w/12,y + h,x–w/4,y + h +10);

line(x + w/12,y + h,x + w/4,y + h +10); }

}

Ćwiczenie 6-7: Dodaj coś do swojego projektu za pomocą pętli for lub while. Czy jest coś, co już masz, a które może być wydajniejsze dzięki pętli?

Ćwiczenie 6-8: Utwórz siatkę kwadratów (każdy kolor losowo) za pomocą pętli for. (Podpowiedź: Będziesz potrzebował dwóch pętli!) Przekoduj ten sam wzorzec za pomocą pętli "while" zamiast "for"

Projekt Numer Dwa

Krok 1. Wybierz projekt lekcji i przepisz go za pomocą zmiennych zamiast wartości zakodowanych na stałe. Rozważ użycie pętli for podczas tworzenia projektu.

Krok 2. Napisz serię operacji przypisania, które zmieniają wartości tych zmiennych i zmieniają wygląd projektu. Możesz także użyć zmiennych systemowych, takich jak width, height, mouseX i mouseY.

Krok 3. Używanie instrukcji warunkowych zmienia zachowanie projektu na podstawie określonych warunków. Co się stanie, jeśli dotknie krawędzi ekranu lub zwiększy się do określonego rozmiaru? Co się stanie, jeśli przesuniesz mysz nad elementami w projekcie?

Użyj przestrzeni podanej poniżej, aby szkicować projekty, notatki i pseudokod dla projektu.

VII FUNKCJE

7.1 Rozbij to

Przykłady przedstawione w częściach od 1 do 6 są krótkie. Prawdopodobnie nie patrzyliśmy na szkic z ponad 100 liniami kodu. Te programy są odpowiednikiem napisania pierwszego akapitu tej części w przeciwieństwie do całej części.

Processing jest wspaniały, ponieważ możemy tworzyć ciekawe szkice wizualne z niewielkimi ilościami kodu. Ale gdy posuniemy się naprzód, by przyjrzeć się bardziej złożonym projektom, takim jak aplikacje sieciowe lub programy do przetwarzania obrazu, zaczniemy mieć setki linii kodu. Będziemy pisać eseje, a nie paragrafy. A te duże ilości kodu mogą okazać się nieporęczne w naszych dwóch głównych blokach - setup() i draw(). Funkcje są sposobem na przejęcie części naszego programu i podzielenie ich na części modułowe, dzięki czemu nasz kod jest łatwiejszy do odczytania, a także do zmiany. Rozważmy grę Space Invaders. Nasze kroki dla funkcji draw() mogą wyglądać mniej więcej tak:

• Wymaż tło.

• Narysuj statek kosmiczny.

• Wyciągaj wrogów.

• Przesuń statek kosmiczny zgodnie z interakcją z klawiaturą użytkownika.

• Poruszaj wrogów.

Co z nazwą?

Funkcje są często nazywane innymi nazwami, takimi jak "Procedury" lub "Metody" lub "Podprogramy". W niektórych językach programowania istnieje rozróżnienie między procedurą (wykonuje zadanie) a funkcją (oblicza wartość). Tu wybieram użycie terminu "funkcja" ze względu na prostotę. Niemniej jednak terminem technicznym w języku programowania Java jest "metoda" (związana z projektowaniem obiektowym Javy), a gdy przejdziemy do obiektów w części 8, będziemy używać terminu "metoda" do opisywania funkcji wewnątrz obiektów.

Przed tą częścią na temat funkcji, przetłumaczylibyśmy powyższy pseudokod na rzeczywisty kod, i umieściłem go wewnątrz draw (). Funkcje jednak pozwolą nam podejść do problemu w następujący sposób:

```
void draw() {

background(0);  // Wywołujemy funkcje, które stworzyliśmy wewnątrz funkcji draw ()!

drawSpaceShip();

drawEnemies();

moveShip();

moveEnemies();

}
```

Powyższe pokazuje, w jaki sposób funkcje ułatwią nam życie dzięki jasnemu i łatwemu do zarządzania kodowi. Niemniej jednak brakuje nam ważnego elementu: definicji funkcji. Wywołanie funkcji to stary

kapelusz. Robimy to cały czas, pisząc line(), rect(), fill () i tak dalej. Definiowanie nowej "wymyślonej" funkcji będzie ciężką pracą. Zanim przejdziemy do szczegółów, spójrzmy, dlaczego pisanie naszych własnych funkcji jest tak ważne:

* Modułowość - funkcje dzielą większy program na mniejsze części, dzięki czemu kod jest łatwiejszy do zarządzania i czytania. Na przykład, kiedy już udało nam się narysować statek kosmiczny

weź ten fragment kodu rysunku statku kosmicznego, przechowuj go w funkcji i wywołaj tę funkcję, gdy zajdzie taka potrzeba (nie martwiąc się o szczegóły samej operacji).

* Możliwość ponownego użycia - Funkcje pozwalają nam ponownie użyć kodu bez konieczności ponownego wpisywania go. Co jeśli chcemy stworzyć grę Space Invaders dla dwóch graczy z dwoma statkami kosmicznymi? Możemy ponownie użyć funkcji drawSpaceShip (), wywołując ją wielokrotnie, bez konieczności powtarzania kodu w kółko.

Tu przyjrzymy się niektórym z naszych poprzednich programów, napisanym bez funkcji, oraz wykazujemy moc modułowości i możliwości ponownego wykorzystania poprzez włączenie funkcji. Ponadto będziemy dalej podkreślać różnice między zmiennymi lokalnymi i globalnymi, ponieważ funkcje są niezależnymi blokami kodu, które będą wymagać użycia zmiennych lokalnych. Na koniec będziemy nadal śledzić historię Zooga za pomocą funkcji. Wywołujemy funkcje, które stworzyliśmy wewnątrz funkcji draw ()!

Ćwiczenie 7-1: Napisz odpowiedzi poniżej.

Jakie funkcje możesz napisać w swoim Projekcie "Lekcja druga"?

Jakie funkcje możesz napisać, aby zaprogramować grę Pong?

7.2 Funkcje zdefiniowane przez użytkownika

W Processing używamy funkcji przez cały czas. Kiedy mówimy "linia (0,0,200,200); "Wywołujemy funkcję function (), wbudowaną funkcję środowiska Processing. Zdolność do narysowania linii przez wywołanie funkcji line () nie istnieje w sposób magiczny. Ktoś, gdzieś zdefiniowany (tj. Napisał kod źródłowy), w jaki sposób Przetwarzanie powinno wyświetlać linię. Jedną z mocnych stron Processing jest biblioteka dostępnych funkcji, którą zaczęliśmy badać w pierwszych sześciu rozdziałach tej książki. Teraz czas się ruszyć poza wbudowanymi funkcjami Processing i napisać własne funkcje zdefiniowane przez użytkownika (AKA "zmyślone").

7.3 Definiowanie funkcji

Definicja funkcji (czasami określana jako "deklaracja") składa się z trzech części:

• Zwróć typ.

• Nazwa funkcji.

• Argumenty.

To wygląda tak:

returnType functionName (argumenty) {

// Ciało kodu funkcji

}

Deja vu?

Pamiętasz, kiedy w Części 3 wprowadziliśmy funkcje setup () i draw ()? Zauważ, że mają one ten sam format, którego teraz się uczymy.

setup() i draw() są funkcjami, które definiujemy i są wywoływane automatycznie przez Processing w celu uruchomienia szkicu. Wszystkie inne funkcje, które piszemy, muszą być przez nas wywoływane. Na razie skupmy się wyłącznie na functionName i treści kodu, ignorując "returnType" i "argumenty". Oto prosty przykład:

Przykład 7-1: Definiowanie funkcji

void drawBlackCircle() {

fill(0);

ellipse(50,50,20,20);

}

Jest to prosta funkcja, która wykonuje jedno podstawowe zadanie: rysowanie elipsy w kolorze czarnym na współrzędnych (50,50). Jego nazwa-drawBlackCircle () - jest dowolna (my ją wymyśliliśmy), a jej treść kodu składa się z dwóch instrukcji (możemy mieć tyle lub mniej, ile chcemy). Ważne jest również, aby przypomnieć sobie, że jest to tylko definicja funkcji. Kod nigdy się nie stanie, chyba że funkcja jest faktycznie wywoływana z części programu, która jest wykonywana. T jest osiągane przez odwołanie się do nazwy funkcji, to jest wywołanie funkcji, jak pokazano w Przykładzie 7-2.

Przykład 7-2: Wywoływanie funkcji

void draw() {

background(255);

drawBlackCircle();

}

Ćwiczenie 7-2: Napisz funkcję wyświetlającą Stworka (lub własny projekt). Wywołaj tę funkcję z wewnątrz funkcji draw()

void setup() {

size(200,200);

}

void draw() {

background(0);

_____

}

_____ _____ _____ {

_____

_____

_____

_____

_____

7.4 Prosta modułowość

Przyjrzyjmy się przykładowi podskakującej kulki z rozdziału 5 i przepiszmy ją za pomocą funkcji, ilustrując jedną technikę dzielenia programu na części modułowe. Przykład 5-6 jest przedrukowany tutaj dla twojej wygody.

```
// Declare global variables

int x = 0;

int speed = 1;

void setup() {

size(200,200);

smooth();

}

void draw() {

background(255);

// Change x by speed

x = x + speed;   // Przesuń piłkę!

// If we've reached an edge, reverse speed

if ((x > width) || (x < 0)) {

speed = speed * –1;  // Odbijaj piłkę!

}

// Display circle at x location

stroke(0);

fill(175);  // Pokaż piłkę!

ellipse(x,100,32,32);

}
```

Po ustaleniu, w jaki sposób chcemy podzielić kod na funkcje, możemy pobrać elementy z draw () i wstawić je do definicji funkcji, wywołując te funkcje wewnątrz funkcji draw (). Funkcje zazwyczaj są zapisywane poniżej funkcji draw ().

Przykład 7-3: Odbijająca się piłka z funkcjami

```
// Declare all global variables (stays the same)

int x = 0;

int speed = 1;

// Setup does not change

void setup() {

size(200,200);

smooth();

}

void draw() {

background(255);

move();        // Zamiast pisać cały kod o kuli to draw (), po prostu wywołujemy trzy funkcje. Skąd
               //wiemy nazwy tych funkcji? Zrobiliśmy je!

bounce();

display();

}

// A function to move the ball

void move() {   // Gdzie należy umieścić funkcje?


// Change the x location by speed

x = x + speed;

}

// A function to bounce the ball

void bounce() {  // If we've reached an edge, reverse speed

if ((x > width) || (x < 0)) { // Możesz zdefiniować swoje funkcje w dowolnym miejscu kodu poza setup
() i draw ().

speed = speed * - 1;

}

}
```

```
// A function to display the ball

void display() {

stroke(0);

fill(175);  // Konwencja polega jednak na umieszczeniu definicji funkcji pod rysunkiem ().

ellipse(x,100,32,32);

}
```

Zwróć uwagę, jak proste stało się draw(). Ten kod jest zredukowany do wywołań funkcji; szczegóły dotyczące zmiany zmiennych i kształtów są wyświetlane dla definicji funkcji. Jedną z głównych zalet jest tu rozsądek programisty. Jeśli napisałeś ten program tuż przed wyjazdem na dwutygodniowe wakacje na Karaibach, po powrocie z ładną opaleniznę, zostałbyś powitany przez dobrze zorganizowany, czytelny kod. Aby zmienić sposób renderowania kuli, wystarczy dokonać edycji funkcji display (), bez konieczności przeszukiwania długich fragmentów kodu lub martwienia się o resztę programu. Na przykład spróbuj zastąpić display () następującym:

```
void display() {

background(255);         // Aby zmienić wygląd kształtu, można ponownie napisać funkcję display (),
                         //pozostawiając wszystkie inne elementy szkicu nienaruszone.

rectMode(CENTER);

noFill();

stroke(0);

rect(x,y,32,32);

fi ll(255);

rect(x - 4,y - 4,4,4);

rect(x + 4,y - 4,4,4);

line(x - 4,y + 4,x + 4,y + 4);

}
```

Inną zaletą korzystania z funkcji jest większa łatwość debugowania. Przypuśćmy na chwilę, że nasza funkcja odbijającej się piłki nie zachowuje się właściwie. Aby znaleźć problem, teraz mieć możliwość włączania i wyłączania części programu. Na przykład możemy po prostu uruchomić program tylko z display (), komentując out move () i bounce ():

```
void draw() {

background(0);  // Funkcje można komentować, aby określić, czy powodują błąd lub nie.

// move();

// bounce();

display();

}
```

Wciąż istnieją definicje funkcji move () i bounce (), tylko teraz funkcje nie są wywoływane. Dzięki dodawaniu wywołań funkcji jeden po drugim i każdorazowemu wykonywaniu szkicu łatwiej możemy wydedukować lokalizację problematycznego kodu.

Ćwiczenie 7-3: Weź dowolny program Processing, który napisałeś i zmodyfikuj go za pomocą funkcji, jak wyżej. Użyj poniższego spacji, aby utworzyć listę funkcji, które musisz napisać.

7.5 Argumenty

Zaledwie kilka stron temu powiedzieliśmy "Zignoruj ReturnType i Argumenty. "Zrobiliśmy to, aby ułatwić funkcjonowanie poprzez trzymanie się podstaw. Jednak funkcje mają więcej mocy niż zwykłe rozbijanie programu na części. Jednym z kluczy do odblokowania tych mocy jest pojęcie argumentów ("parametry" AKA). Argumenty to wartości, które są "przekazywane" do funkcji. Możesz myśleć o nich jako o warunkach, w których funkcja powinna działać. Zamiast zwykłego powiedzenia "Przenieś", funkcja może powiedzieć "Przenieś liczbę kroków N", gdzie "N" jest argumentem. Kiedy wyświetlamy elipsę w przetwarzaniu, musimy podać szczegóły dotyczące tej elipsy. Nie możemy po prostu powiedzieć, że narysuj elipsę, musimy powiedzieć, że narysuj elipsę w tym miejscu i z takim rozmiarem. Są to argumenty funkcji ellipse () i napotkaliśmy to w części 1, kiedy nauczyliśmy się wywoływać funkcje po raz pierwszy. Zmieńmy teraz metodę drawBlackCircle () na argument:

```
void drawBlackCircle(int diameter) {

fill(0);  // „diameter" jest argumentem funkcji drawBlackCircle ().

ellipse(50,50, diameter, diameter);

}
```

Argument jest po prostu deklaracją zmiennej wewnątrz nawiasów w definicji funkcji. Th jest zmienna jest lokalną zmienną do użycia w tej funkcji (i tylko w tej funkcji). Białe kółko będzie wymiarowane zgodnie z wartością umieszczoną w nawiasach.

```
drawBlackCircle (16); // Narysuj okrąg o średnicy 16

drawBlackCircle (32); // Narysuj okrąg o średnicy 32
```

Patrząc na przykład podskakującej kuli, moglibyśmy przepisać funkcję move () na argument:

```
void move (int speedFactor) { // Argument "speedFactor" wpływa na szybkość poruszania się koła.

x = x + (prędkość * szybkość);

}
```

Aby przenieść piłkę dwukrotnie szybciej:

```
move(2);
```

Lub 5-krotnie:

```
move(5);
```

Możemy również przekazać do funkcji inną zmienną lub wynik wyrażenia matematycznego (takiego jak mouseX podzielony przez 10). Na przykład:

move (mouseX / 10);

Argumenty torują drogę bardziej elastycznym, a zatem wielokrotnego użytku, funkcjom. Aby to zademonstrować, przyjrzymy się kodowi do rysowania kolekcji kształtów i zbadamy, w jaki sposób funkcje pozwalają nam narysować wiele wersji wzorca bez powtarzania tego samego kodu w kółko. Opuszczając Zooga nieco później, rozważ następujący wzór przypominający samochód (oglądany z góry, jak pokazano na rysunku):

```
size(200,200);

background(255);

int x = 100; // x location

int y = 100; // y location

int thesize = 64; // size

int offset = thesize/4; // position of wheels relative to car

// draw main car body (i.e. a rect)

rectMode(CENTER);

stroke(0);

fill(175);

rect(x,y,thesize,thesize/2);      //Kształt samochodu to pięć prostokątów, jeden duży prostokąt w
                                  //środku i cztery koła na zewnątrz

// draw four wheels relative to center

fill(0);

rect(x – offset,y – offset,offset,offset/2);

rect(x + offset,y – offset,offset,offset/2);

rect(x – offset,y + offset,offset,offset/2);

rect(x + offset,y + offset,offset,offset/2);
```

Aby narysować drugi samochód, powtarzamy powyższy kod z różnymi wartościami, jak pokazano na rysunku

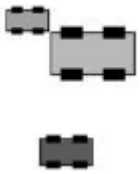
```
x = 50; // x location

y = 50; // y location

thesize = 24; // size

offset = thesize/4; // position of wheels relative to car

// draw main car body (i.e. a rect)

rectMode(CENTER);

stroke(0);

fill(175);

rect(x,y,thesize,thesize/2);

// draw four wheels relative to center  //Każda linia kodu jest powtarzana, aby narysować drugi
                                        //samochód.

fill(0);

rect(x - offset,y - offset,offset,offset/2);

rect(x + offset,y - offset,offset,offset/2);

rect(x - offset,y + offset,offset,offset/2);

rect(x + offset,y+offset,offset,offset/2);
```

Powinno być dość oczywiste, gdzie to się dzieje. W końcu robimy to samo dwa razy, po co zawracać sobie głowę powtarzaniem całego tego kodu? Aby uniknąć tego powtórzenia, możemy przenieść kod do funkcji wyświetlającej samochód zgodnie z kilkoma argumentami (pozycja, rozmiar i kolor).

```
void drawcar(int x, int y, int thesize, color c) {

// Using a local variable "offset"

int offset = thesize/4; //Zmienne lokalne mogą być zadeklarowane i użyte w funkcji!

// Draw main car body

rectMode(CENTER);

stroke(200);

fill(c);

rect(x,y,thesize,thesize/2);

// Draw four wheels relative to center

fill(200);        //Ten kod jest definicją funkcji. Funkcja drawCar () rysuje kształt samochodu w oparciu
                  // o cztery argumenty: położenie w poziomie, położenie w pionie, rozmiar i kolor.

rect(x - offset,y - offset,offset,offset/2);

rect(x + offset,y - offset,offset,offset/2);
```

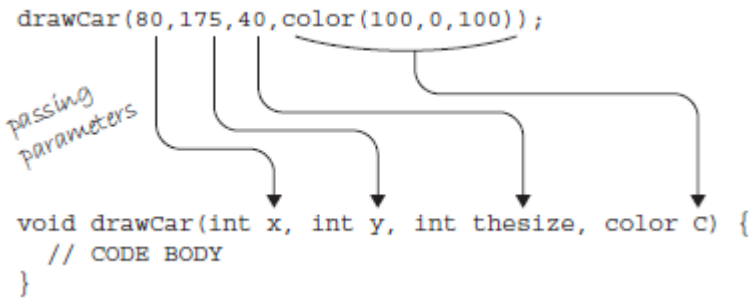
```
rect(x - offset,y + offset,offset,offset/2);

rect(x + offset,y + offset,offset,offset/2);

}
```

W funkcji draw () wywołujemy wówczas funkcję drawCar () trzy razy, za każdym razem przekazując cztery parametry. Zobacz dane wyjściowe na rysunku



```
void setup() {

size(200,200);

}

void draw() {

background(0);  // Ten kod wywołuje funkcję trzy razy, z dokładną liczbą parametry we właściwej
                //kolejności.

drawCar(100,100,64,color(200,200,0));

drawCar(50,75,32,color(0,200,100));

drawCar(80,175,40,color(200,0,0));

}
```

Technicznie rzecz biorąc, argumenty są zmiennymi, które żyją wewnątrz nawiasów w definicji funkcji, tj. "Void drawCar (int x, int y, int thesize, color c). "Parametry są wartościami przekazanymi do funkcji, gdy jest wywoływana, czyli "drawCar (80,175,40, kolor (100,0,100)); ". Różnica semantyczna między argumentami i parametrami jest dość trywialna i nie powinniśmy się przejmować, jeśli od czasu do czasu pomylimy użycie tych dwóch słów. Koncepcją, na której należy się skoncentrować, jest umiejętność przekazywania parametrów. Nie będziemy w stanie pogłębić naszej wiedzy programistycznej, jeśli nie będziemy się dobrze czuli z tą techniką. Przejdźmy z przepustką słów. Wyobraź sobie piękny, słoneczny dzień i baw się z przyjacielem w parku. Masz piłkę. Ty (główny program) zadzwoń do funkcji (twój przyjaciel) i podaj piłkę (argument). Twój znajomy (funkcja) ma teraz piłkę (argument) i może jej użyć, ale jej się podoba (sam kod w funkcji)



Ważne rzeczy do zapamiętania na temat przekazywania parametrów

* Musisz podać taką samą liczbę parametrów, jak zdefiniowano w funkcji.

* Gdy parametr jest przekazywany, musi być tego samego typu, co zadeklarowany w argumentach w definicji funkcji. Liczba całkowita musi być przekazana do liczby całkowitej, zmiennoprzecinkowa do punktu zmiennoprzecinkowego i tak dalej.

* Wartość, którą przekazujesz jako parametr funkcji, może być wartością literalną (20, 5, 4.3 itd.), Zmienną (x, y itd.) Lub wynikiem wyrażenia (8 + 3, 4 * x / 2, losowe (0,10), itp.)

* Argumenty działają jako zmienne lokalne dla funkcji i są dostępne tylko w ramach tej funkcji.

Ćwiczenie 7-4: Następująca funkcja pobiera trzy liczby, dodaje je do siebie i drukuje sumę do okna wiadomości.

```
void sum(int a, int b, int c) {

int total = a + b + c;

println(total);

}
```

Patrząc na powyższą definicję funkcji, napisz kod, który wywołuje funkcję.

_____

Ćwiczenie 7-5: OK, tutaj jest przeciwny problem. Oto linia kodu, który zakłada funkcję, która pobiera dwie liczby, mnoży je razem i drukuje wynik do okna komunikatu. Napisz definicję funkcji związaną z tym wywołaniem funkcji.

```
multiply(5.2,9.0);
```

_____

_____

_____

_____

Ćwiczenie 7-6: Oto odbijająca kula z przykładu 5-6 połączona z funkcją drawCar (). Wypełnij puste miejsca, aby mieć teraz samochód podskakujący z parametryzacją! (Zwróć uwagę, że globalne zmienne mają teraz nazwę globalX i globalY, aby uniknąć pomyłki ze zmiennymi lokalnymi x i y w drawCar ()).

```
int globalX = 0;

int globalY = 100;

int speed = 1;

void setup() {

size(200,200);
```

```
smooth();
}
void draw() {
background(0);
_____
_____
_____
}
void move() {
// Change the x location by speed
globalX = globalX + speed;
}
void bounce() {
if ((globalX > width) || (globalX < 0)) {
speed = speed * –1;
}
}
void drawCar(int x, int y, int thesize, color c) {
int offset = thesize / 4;
rectMode(CENTER);
stroke(200);
fill(c);
rect(x,y,thesize,thesize/2);
fill(200);
rect(x - offset,y - offset,offset,offset/2);
rect(x + offset,y - offset,offset,offset/2);
rect(x - offset,y + offset,offset,offset/2);
rect(x + offset,y + offset,offset,offset/2);
}
```

7.6 Przekazywanie kopii

Pojawił się niewielki problem z analogią "grających haczyków". To, co powinienem powiedzieć, to: Przed podrzuceniem piłki (argumentem), robisz jej kopię (druga kulka) i przekazujesz ją do odbiorcy (funkcja). Za każdym razem, gdy przekazujesz do funkcji prymitywną wartość (integer, fl at ow, char, itp.), Nie przekazujesz samej wartości, lecz kopię tej zmiennej. To może wydawać się trywialnym rozróżnieniem, gdy podaje się zakodowaną liczbę, ale nie jest tak trywialne, gdy przekazujemy zmienną. Poniższy kod ma funkcję o nazwie randomizer (), która otrzymuje jeden argument (liczba zmiennoprzecinkowa) i dodaje liczbę losową między - 2 i 2 do niego. Oto pseudokod.

* num to liczba 10.

* wyświetlana liczba: 10

* Kopia num zostaje przekazana do argumentu newnum w funkcji randomizer ().

* W funkcji randomizer ():

- losowa liczba jest dodawana do newnum.

- wyświetlany jest komunikat newnum: 10.34232

* ponownie wyświetlana jest liczba: nadal 10! Kopia została wysłana do newnum, więc num się nie zmienił.

A oto kod:

```
void setup() {

float num = 10;

println( " The number is: " + num);

randomizer(num);

println( " The number is: " + num);

}

void randomizer(float newnum) {

newnum = newnum + random( – 2,2);

println( " The new number is: " + newnum);

}
```

Mimo że zmienna num została przekazana do zmiennej newnum, która następnie szybko zmieniła wartości, pierwotna wartość zmiennej num nie została naruszona, ponieważ została wykonana kopia. Chciałbym odnieść się do tego procesu jako "przekazywać za pomocą kopii", jednak jest on częściej określany jako "przekazywanie według wartości". "Dotyczy to wszystkich prymitywnych typów danych (jedyne znane nam dotychczas typy: liczba całkowita, liczba klatek itp.), Ale nie będzie tak, gdy dowiadujemy się o obiektach w następnej części. Przykład ten daje nam również przyjemną okazję do przejrzenia przepływu programu podczas korzystania z funkcji. Zauważ, jak kod jest wykonywany w kolejności, w której zapisywane są linie, ale gdy wywoływana jest funkcja, znak kod pozostawia aktualną linię, wykonuje linie wewnątrz funkcji, a następnie wraca do miejsca, w którym została przerwana. Oto opis przepływu powyższego przykładu:

1. Ustaw liczbę równą 10.

2. Wyświetl wartość num.

3. Wywołaj funkcję randomizer.

a. Ustaw newnum równe newnum plus losowa liczba.

b. Wyświetl wartość newnum.

4. Wyświetl wartość num.

Ćwiczenie 7-7: Przewidź wyjście tego programu, wypisując to, co pojawi się w oknie komunikatu.

```
void setup() {
println( "a");
function1();
println( "b");
}
void draw() {
println( "c");
function2();
println( "d");
function1();
noLoop();       //Nowy! noLoop () jest wbudowaną funkcją przetwarzania, która zatrzymuje pętlę
                //draw (). W tym przypadku możemy go użyć, aby upewnić się, że funkcja draw ()
                //wykonuje tylko jeden raz. Możemy go ponownie uruchomić w innym punkcie kodu,
                //wywołując funkcję loop ().
}
void function1() {
println( "e");
println( "f");
}
void function2() {
println( "g");
function1();    //Całkowicie rozsądne jest wywoływanie funkcji z funkcji. W rzeczywistości robimy to
                //cały czas za każdym razem, gdy wywołujemy jakąkolwiek funkcję z wnętrza setup ()
                //lub draw ().
println( "h");
}
```

WYJŚCIE :

7.7 Typ Return

Do tej pory widzieliśmy, jak funkcje mogą oddzielać szkic na mniejsze części, a także włączać argumenty, aby można je było ponownie wykorzystać. Jednak z tej dyskusji brakuje jednego elementu, a jest nim odpowiedź na pytanie, z którym od początku się zastanawiasz: "Co oznacza pustka? "Dla przypomnienia zbadajmy strukturę definicji funkcji:

ReturnType FunctionName (Arguments) {

// Ciało kodu funkcji

}

OK, teraz spójrzmy na jedną z naszych funkcji:

// Funkcja do poruszania piłką

void move (int speedFactor) {

// Zmień położenie x organizmu przez prędkość pomnożoną przez speedFactor

x = x + (prędkość * szybkość);

}

"Move" to FunctionName, "speedFactor" jest argumentem dla funkcji, a "void" to ReturnType. Wszystkie funkcje, które zdefiniowaliśmy do tej pory, nie miały typu zwrotu; właśnie to oznacza "pustka": brak typu zwrotu. Ale jaki jest typ zwrotu i kiedy możemy go potrzebować? Przypomnijmy sobie przez chwilę funkcję random (), którą sprawdziliśmy w części 4. Zapytaliśmy o funkcję dla liczby losowej między 0 a pewną wartością, a random () łaskawie potraktowali naszą prośbę i zwrócili nam losową wartość w odpowiednim zakresie. Funkcja random () zwróciła wartość. Jaki typ wartości? Liczba zmiennoprzecinkowa. W przypadku random(), jego typem zwracanym jest float. Typ zwracany jest typem danych zwracanym przez funkcję. W przypadku random() nie określiliśmy, zwróć jednak uwagę , że twórcy Processing zrobili i jest to udokumentowane na stronie referencyjnej dla random(). Za każdym razem, gdy wywoływana jest funkcja random (), zwraca nieoczekiwaną wartość w podanym zakresie. Jeśli jeden parametr zostanie przekazany do funkcji, zwróci strumień o wartości od zera do wartości parametru. Wywołanie funkcji random (5) zwraca wartości między 0 a 5. Jeśli zostaną przekazane dwa parametry, zwróci wartość float z wartością między parametrami. Funkcja losowa (-5, 10.2) zwraca wartości od -5 do 10.2. -Z http://www.processing.org/reference/random.html . Jeśli chcemy napisać własną funkcję, która zwraca wartość, musimy określić typ w definicji funkcji. Stwórzmy prosty przykład:

int sum(int a, int b, int c) { // Ta funkcja, która dodaje trzy liczby razem, ma typ zwrotu - int.

int total = a + b + c;

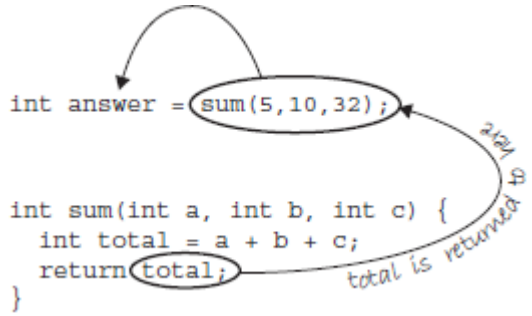
return total; //Wymagana jest instrukcja return! Funkcja z typem zwracanym musi zawsze zwracać
//wartość tego typu

}

Zamiast zapisywania pustki jako typu zwrotu, jak to mamy w poprzednich przykładach, teraz piszemy int. Th określa, że funkcja musi zwracać wartość typu integer. Aby funkcja zwróciła wartość, wymagana jest instrukcja return. Instrukcja return wygląda następująco:

return valueToReturn;

Jeśli nie uwzględnimy zwrotu, Przetwarzanie dałoby nam błąd: Th e metoda "int sum (int a, int b, int c); "Musi zawierać instrukcję return z wyrażeniem zgodnym z typem" int ". "Gdy tylko instrukcja return zostanie wykonana, program opuszcza funkcję i wysyła zwróconą wartość z powrotem do lokalizacji w kodzie, w którym wywołano tę funkcję. Ta wartość może być używana w zadaniu operacja (w celu nadania innej zmiennej wartości) lub w dowolnym odpowiednim wyrażeniu. Zobacz ilustrację na rysunku.



Oto kilka przykładów:

int x = sum(5,6,8);

int y = sum(8,9,10) * 2;

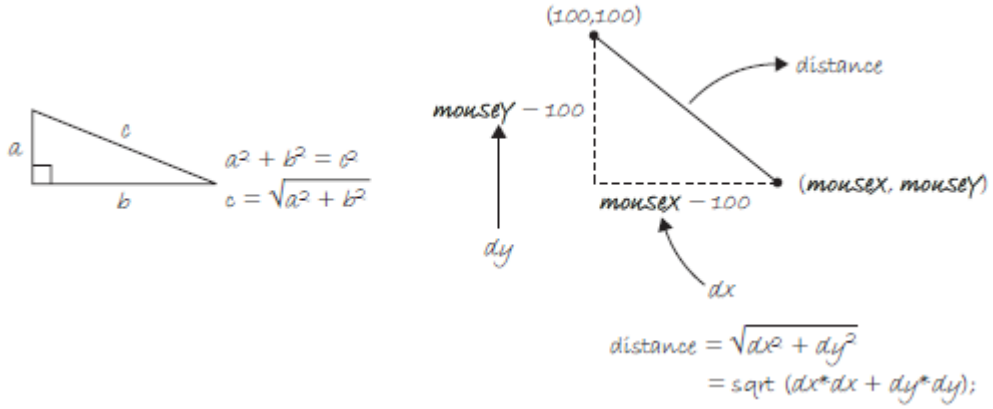
int z = sum(x,y,40);

line(100,100,110,sum(x,y,z));

Nienawidzę wychodzić z haczyka w parku, ale możesz myśleć o tym w następujący sposób. Ty (główny program) rzucasz piłkę do swojego przyjaciela (funkcja). Po tym, jak twój przyjaciel łapie piłkę, on lub ona myśli przez chwilę, umieszcza liczbę w kuli (wartość zwrotu) i przekazuje ją z powrotem do ciebie. Funkcje, które zwracają wartości, są tradycyjnie używane do wykonywania złożonych obliczeń, które mogą wymagać wykonywania wielokrotnie w trakcie trwania programu. Jednym z przykładów jest obliczenie odległości między dwoma punktami: (x 1, y 1) i (x 2, y 2). Odległość między pikselami jest bardzo użyteczną informacją aplikacje interaktywne. Przetwarzanie ma wbudowaną funkcję odległości, z której możemy korzystać. Nazywa się dist().

float d = dist (100, 100, mouseX, mouseY); //Obliczanie odległości między (100,100) i (mouseX,
//mouseY).

Ten wiersz kodu oblicza odległość między lokalizacją myszy a punktem (100,100). Na razie udawajmy, że Processing nie zawarł tej funkcji w swojej bibliotece. Bez tego musielibyśmy obliczyć odległość ręcznie, korzystając z Twierdzenia Pitagorasa, jak pokazano na Rysunku



float dx = mouseX - 100;

float dy = mouseY - 100;

float d = sqrt (dx * dx + dy * dy);

Gdybyśmy chcieli wykonać to obliczenie wiele razy w trakcie programu, łatwiej byłoby przenieść go do funkcji, która zwraca wartość d.

float distance(float x1, float y1, float x2, float y2) {

float dx = x1 – x2; // Nasza wersja funkcji dist () Processing.

float dy = y1 – y2;

float d = sqrt(dx*dx + dy*dy);

return d;

}

Zwróć uwagę na użycie pływającego typu powrotu. Ponownie, nie musimy pisać tej funkcji, ponieważ Processing dostarcza ją dla nas. Ale odkąd to zrobiliśmy, możemy teraz spojrzeć na przykład, który korzysta z tej funkcji.

Przykład 7-4: Używanie funkcji, która zwraca wartość, odległość

void setup() {

size(200,200);

}

void draw() {

background(0);

stroke(0);

// Top left square // Nasza funkcja odległości służy do obliczania wartości jasności dla każdego //kwadrantu. Zamiast tego możemy użyć wbudowanej funkcji dist (), ale //uczymy się pisać własne funkcje.

fill(distance(0,0,mouseX,mouseY));

```
rect(0,0,width/2 – 1,height/2 – 1);
// Top right square
fill(distance(width,0,mouseX,mouseY));
rect(width/2,0,width/2 – 1,height/2 – 1);
// Bottom left square
fill(distance(0,height,mouseX,mouseY));
rect(0,height/2,width/2 – 1,height/2 – 1);
// Bottom right square
fill(distance(width,height,mouseX,mouseY));
rect(width/2,height/2,width/2 – 1,height/2 – 1);
}
float distance(float x1, float y1, float x2, float y2)
{
float dx = x1 – x2;
float dy = y1 – y2;
float d = sqrt(dx*dx + dy*dy);
return d;
}
```

Ćwiczenie 7-8: Napisz funkcję, która wymaga jednego argumentu F dla Fahrenheita i obliczyć wynik następującego równania (przeliczając temperaturę na Celsjusza).

C = (F - 32) * (5/9)

```
_____ tempConverter (float _________) {
_____ ________ = _________________
_____________________________________
}
```

7.8 Reorganizacja Stworka

Stworek jest teraz gotowy na gruntowny remont.

• Reorganizacja Zooga za pomocą dwóch funkcji: drawZoog () i jiggleZoog (). Tylko dla odmiany, będziemy się poruszać Zoogem (poruszaj się losowo w obu kierunkach), zamiast odbijać się w przód i w tył.

• Włącz argumenty, aby zgrzyt Zooga był określony przez pozycję myszyX, a kolor oczu Zooga jest określony przez odległość Zooga od myszy.

Przykład 7-5: Stworek z funkcjami

```
float x = 100;
float y = 100;
float w = 60;
float h = 60;
float eyeSize = 16;
void setup() {
size(200,200);
smooth();
}
void draw() {
background(255); // Draw a black background
// mouseX position determines speed factor for moveZoog function
// float factor = constrain(mouseX/10,0,5);
jiggleZoog(factor);
// pass in a color to drawZoog   //Kod do zmiany zmiennych związanych ze Stworkiem i wyświetlaniem
                                 //Stworka  jest przenoszony poza funkcje draw () i do funkcji tutaj.
                                 //Funkcje są podane argumenty, takie jak "Jiggle Zoog przez
                                 //następujący czynnik" i "narysuj Stworkag z następującym kolor
                                 //oczu."
// function for eye's color
float d = dist(x,y,mouseX,mouseY);
color c = color(d);
drawZoog(c);
}
void jiggleZoog(float speed) {
// Change the x and y location of Zoog randomly
x = x + random( - 1,1)*speed;
y = y + random( - 1,1)*speed;
// Constrain Zoog to window
x = constrain(x,0,width);
y = constrain(y,0,height);
```

```
}
void drawZoog(color eyeColor) {
// Set ellipses and rects to CENTER mode
ellipseMode(CENTER);
rectMode(CENTER);
// Draw Zoog's arms with a for loop
for (float i = y - h/3; i < y + h/2; i + = 10) {
stroke(0);
line(x - w/4,i,x + w/4,i);
}
// Draw Zoog's body
stroke(0);
fill(175);
rect(x,y,w/6,h);
// Draw Zoog's head
stroke(0);
fill(255);
ellipse(x,y - h,w,h);
// Draw Zoog's eyes
fill(eyeColor);
ellipse(x - w/3,y - h,eyeSize,eyeSize*2);
ellipse(x + w/3,y - h,eyeSize,eyeSize*2);
// Draw Zoog's legs
stroke(0);
line(x - w/12,y + h/2,x - w/4,y + h/2 + 10);
line(x + w/12,y + h/2,x + w/4,y + h/2 + 10);
}
```

Ćwiczenie 7-9: Poniżej znajduje się wersja przykładu 6-11 ("wiele Stworków"), która wywołuje funkcję narysowania Stworka. Napisz definicję funkcji, która kończy ten szkic. W trakcie tego procesu możesz przeprojektować Stworka

```
void setup() {
size(400,200); // Set the size of the window
```

```
smooth(); // Enables Anti-Aliasing (smooth edges on
shapes)
}
void draw() {
background(0); // Draw a black background
int y = height/2;
// Multiple versions of Zoog are displayed by using a for loop
for (int x = 80; x < width; x + = 80) {
drawZoog(x,100,60,60,16);
}
}
```

_____

_____

_____

_____

_____

_____

_____

_____

Ćwiczenie 7-10: Przepisz Projekt Lekcji Drugiej przy użyciu funkcji. Jeszcze 10! Kopia została wysłana do newnum, więc num się nie zmienił.

VIII Obiekty

8.1 Zabrakło mi OOP.

Zanim zaczniemy analizować szczegóły tego, jak programowanie obiektowe (OOP) działa w Processing, przejdźmy do krótkiej konceptualnej dyskusji o samych "obiektach". Ważne jest, aby zrozumieć, że nie wprowadzamy żadnych nowych podstaw programowania: obiekty wykorzystują wszystko, czego już się nauczyliśmy: zmienne, wyrażenia warunkowe, pętle, funkcje i tak dalej. Jednak zupełnie nowym jest sposób myślenia, sposób uporządkowania i uporządkowania wszystkiego, czego już się nauczyliśmy. Wyobraź sobie, że nie programujesz w Processing, ale zamiast tego napisałeś program na swój dzień, listę instrukcji, jeśli chcesz. Może zacząć się coś takiego:

• Obudź się.

• Pij kawę (lub herbatę).

• Jedz śniadanie: płatki, jagody i mleko sojowe.

• Jeździć po metrze.

Co tu się dzieje? Specyficznie, jakie rzeczy są w to zaangażowane? Po pierwsze, chociaż może nie być od razu widoczne, jak napisaliśmy powyższe instrukcje, najważniejsze jest to, że jesteś człowiekiem, osobą. Wystawiasz pewne właściwości. Wyglądasz w określony sposób; może masz brązowe włosy, nosisz okulary i wyglądasz trochę nerwowo. Masz także możliwość robienia rzeczy, takich jak budzenie (prawdopodobnie możesz też spać), jedzenie lub jazda metrem. Obiekt jest taki jak Ty, rzeczą, która ma właściwości i może robić rzeczy. Jak to się ma do programowania? Właściwości obiektu to zmienne; a rzeczy, które obiekt może zrobić, są funkcjami. Programowanie obiektowe to połączenie wszystkiego, czego nauczyliśmy się w częściach od 1 do 7, danych i funkcjonalności, a wszystko to w jedną całość. Przekreślmy dane i funkcje bardzo prostego obiektu ludzkiego:

<u>Dane ludzkie</u>

• Wysokość.

• Waga.

• Płeć.

• Kolor oczu.

• Kolor włosów.

<u>Funkcje ludzkie</u>

• Sen .

• Obudź się.

• Jeść .

• Jedź jakąś formą transportu.

Teraz, zanim przejdziemy za daleko, musimy rozpocząć krótką metafizyczną dygresję. Powyższa struktura nie jest samym człowiekiem; po prostu opisuje ideę, czyli pojęcie, za człowiekiem. Opisuje to, czym jest być człowiekiem. Być człowiekiem to mieć wzrost, włosy, spać, jeść i tak dalej. Jest to kluczowe wyróżnienie dla obiektów programistycznych. Ten szablon człowieka jest znany jako klasa.

Klasa różni się od obiektu. Jesteś obiektem. Jestem przedmiotem. Chłop w metrze jest obiektem. Albert Einstein jest obiektem. Wszyscy jesteśmy ludźmi, rzeczywistymi przykładami idei iistota ludzka. Pomyśl o foremce do ciastek. Obcinacz plików cookie tworzy pliki cookie, ale nie jest plikiem cookie. Ta przecinarka jest klasą, ciasteczka są obiektami.

Ćwiczenie 8-1: Rozważ samochód jako obiekt. Jakie dane miałby samochód? Jakie funkcje miałaby ta funkcja?

8.2 Używanie obiektu

Zanim przejrzymy faktyczne pisanie samej klasy, spójrzmy, jak używanie obiektów w naszym głównym programie (tj. Setup () i draw ()) sprawia, że świat staje się lepszym miejscem. Wracając do przykładu samochodu z rozdziału 7, możesz przypomnieć sobie, że wyglądał pseudokod dla szkicu coś takiego:

Dane (zmienne globalne):

Kolor samochodu.

Samochód x lokalizacja.

Lokalizacja samochodu.

Samochód x prędkość.

Ustawiać:

Inicjalizuj kolor samochodu.

Zainicjuj lokalizację samochodu na punkt początkowy.

Inicjalizuj prędkość samochodu.

Rysować:

Wypełnij tło.

Wyświetl samochód w lokalizacji z kolorem.

Przyspiesz lokalizację samochodu.

W części 7 zdefiniowaliśmy globalne zmienne u góry programu, zainicjowaliśmy je w setup () i wywołaliśmy funkcje do poruszania się i wyświetlania samochodu w trybie draw (). Programowanie obiektowe pozwala nam usunąć wszystkie zmienne i funkcje z programu głównego i zapisać je w obiekcie samochodowym. Obiekt samochodowy będzie wiedział o jego danych - kolorze, lokalizacji, prędkości. To jest część pierwsza. Część druga obiektu samochodu to rzeczy, które może zrobić, metody (funkcje wewnątrz obiektu). Samochód może się poruszać i można go wyświetlić. Przy projektowaniu zorientowanym obiektowo, pseudokod poprawia wygląd w następujący sposób:

Dane (zmienne globalne):

Obiekt samochodowy.

Ustawiać:

Zainicjuj obiekt samochodu.

Rysować:

Wypełnij tło.

Wyświetl obiekt samochodu.

Przesuń obiekt samochodu.

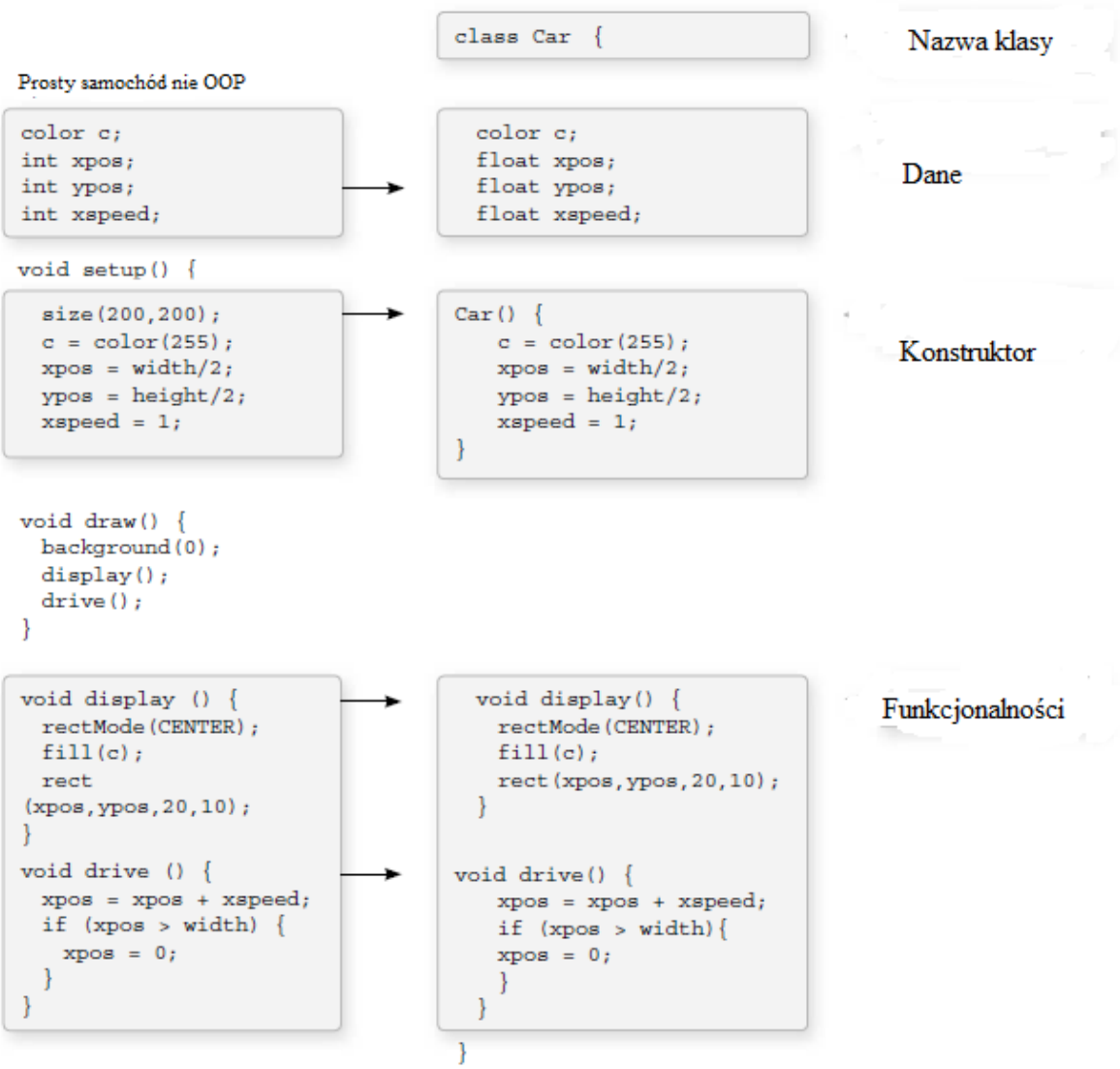
Zauważ, że usunęliśmy wszystkie zmienne globalne z pierwszego przykładu. Zamiast oddzielnych zmiennych dotyczących koloru samochodu, lokalizacji samochodu i prędkości samochodu, mamy teraz tylko jedną zmienną, zmienną samochodu! Zamiast inicjować te trzy zmienne, inicjujemy jedną rzecz, obiekt Car. Gdzie podziały się te zmienne? Nadal istnieją, tylko teraz żyją wewnątrz obiektu Car (i zostaną zdefiniowane w klasie Car, do której dojdziemy za chwilę). Wychodząc poza pseudokod, rzeczywiste ciało szkicu może wyglądać tak:

```
car myCar;   // Obiekt w Processing

void setup() {

myCar = new Car();

}

void draw() {

background(0);

myCar.move();

myCar.display();

}
```

Za chwilę zajmiemy się szczegółami poprzedniego kodu, ale zanim to zrobimy, przyjrzyjmy się, jak napisano klasę samochodów.

8.3 Pisanie szablonowe

Powyższy prosty przykład pokazuje, w jaki sposób użycie obiektu w przetwarzaniu tworzy czysty, czytelny kod. Ciężka praca polega na napisaniu szablonu obiektu, czyli samej klasy. Kiedy uczysz się najpierw programowania obiektowego, często dobrym ćwiczeniem jest wykonanie programu napisanego bez obiektów i, nie zmieniając w ogóle funkcjonalności, przepisać go za pomocą obiektów. Zrobimy to dokładnie na przykładzie samochodu z części 7, odtwarzając dokładnie ten sam wygląd i zachowanie w sposób zorientowany na obiekt. I na końcu tej części przerobimy Stworek jako obiekt. Wszystkie klasy muszą zawierać cztery elementy: nazwę, dane, konstruktor i metody. (Technicznie, jedynym faktycznym wymaganym elementem jest nazwa klasy, ale celem programowania obiektowego jest uwzględnienie wszystkich tych elementów.) .Oto w jaki sposób możemy pobrać elementy z prostego szkicu nieobiektywizowanego (uproszczona wersja rozwiązania do ćwiczenia 7-6) i umieścić je w klasie samochodów, z której następnie będziemy mogli tworzyć obiekty Car



. * Nazwa klasy - nazwa jest określona przez "class WhateverNameYouChoose". Następnie zawieramy cały kod klasy wewnątrz nawiasów klamrowych po deklaracji nazwy. Nazwy klas są tradycyjnie pisane wielką literą (w celu odróżnienia ich od nazw zmiennych, które tradycyjnie są małymi literami).

* Dane - dane dla klasy to zbiór zmiennych. Te zmienne są często określane jako zmienne instancji, ponieważ każda instancja obiektu zawiera ten zbiór zmiennych.

* Konstruktor - Konstruktor jest specjalną funkcją wewnątrz klasy, która tworzy instancję samego obiektu. Jest tam, gdzie podajesz instrukcje, jak ustawić obiekt. Jest to dokładnie tak samo jak funkcja Setup () Processing, tylko tutaj służy do tworzenia pojedynczego obiektu w szkicu, ilekroć tworzony jest nowy obiekt z tej klasy. Zawsze ma taką samą nazwę jak klasa i jest wywoływana przez wywołanie nowego operatora: "Car myCar? nowe auto( ); ".

* Funkcjonalność - możemy dodać funkcjonalność do naszego obiektu, pisząc metody. Robi się to w taki sam sposób, jak opisano w części 7, z typem zwracania, nazwą, argumentami i treścią kodu.

Ten kod dla klasy istnieje jako własny blok i można go umieścić w dowolnym miejscu poza setup () i draw ().

Klasa to nowy blok kodu!

```
void setup () {

}

void draw () {

}

class Car {

}
```

Ćwiczenie 8-2: Wypełnij puste pola poniższą definicją klasy ludzkiej. Włącz funkcję o nazwie sleep () lub utwórz własną funkcję. Postępuj zgodnie ze składnią przykładu Car. (Nie ma dobrych ani złych odpowiedzi pod względem samego kodu, istotna jest struktura).

```
_________ {

color hairColor;

float height;

_________() {

_____________________________

_____________________ _______

}

_____________________________ {

_____________________________

_____________________________

}

}
```

8.4 Używanie obiektu: Szczegóły

W punkcie 8.2 przyjrzeliśmy się, jak obiekt może znacznie uprościć główne części przetwarzania

sketch (setup( ) and draw( )).

```
Car myCar;   // Krok1. Deklarujemy obiekt

void setup() {

myCar = new Car();  // Krok 2. Inicjujemy obiekt

}

void draw() {

background(0);  // Krok 3. Wywołanie metod na obiekcie

myCar.move();

myCar.display();
```

}

Przyjrzyjmy się szczegółom, jakie kryją się za powyższymi trzema krokami, opisującymi sposób użycia obiektu w szkicu.

Krok 1. Deklarowanie zmiennej obiektu.

Jeśli powrócisz do rozdziału 4, możesz pamiętać, że zmienna jest zadeklarowana przez podanie typu i nazwy.

// Deklaracja zmiennej

int var; // Wpisz imię

Powyższe jest przykładem zmiennej, która zawiera prymitywną, w tym przypadku liczbę całkowitą. Jak dowiedzieliśmy się w Części 4, prymitywne typy danych są pojedynczymi jednostkami informacji: liczbą całkowitą, flemą, znakiem. Deklaracja zmiennej, która mieści się na obiekcie, jest dość podobna. Różnica polega na tym, że tutaj typ to nazwa klasy, coś, co uzupełnimy, w tym przypadku "Samochód. "Nawiasem mówiąc, obiekty nie są prymitywami i są uważane za złożone typy danych. (Dzieje się tak dlatego, że przechowują wiele informacji: dane i funkcje. Prymitywy przechowują tylko dane.)

Krok 2. Inicjowanie obiektu.

Ponownie, możesz przypomnieć sobie z częścu 4, że aby zainicjować zmienną (tj. nadać jej wartość początkową), używamy operacji przypisania - zmienna równa się coś.

// Inicjalizacja zmiennych

var = 10; // var równa się 10

Inicjalizacja obiektu jest nieco bardziej złożona. Zamiast po prostu przypisywać mu prymitywną wartość, taką jak liczba całkowita lub liczba zmiennoprzecinkowa, musimy skonstruować obiekt. Obiekt jest tworzony z nowym operatorem.

// Inicjalizacja obiektu

myCar = new Car (); // operator nowy jest używany do zrobienia obiektu

W powyższym przykładzie "myCar" jest nazwą zmiennej obiektu i "? "Wskazuje, że ustawiamy to jako coś, coś, co jest nowym przykładem obiektu Car. To, co tak naprawdę tutaj robimy, to inicjowanie obiektu Car. Po zainicjowaniu zmiennej pierwotnej, takiej jak liczba całkowita, wystarczy ustawić ją równą liczbie. Ale obiekt może zawierać wiele części danych. Przywołując klasę Car z poprzedniej sekcji, widzimy, że ten wiersz kodu wywołuje konstruktor, specjalną funkcję o nazwie Car (), która inicjuje wszystkie zmienne obiektu i upewnia się, że obiekt Car jest gotowy do pracy. Jeszcze jedna rzecz; z pierwotną liczbą całkowitą "var", jeśli zapomniałeś zainicjować ją (ustawiając ją równą 10), przetwarzanie przypisałoby mu wartość domyślną, zero. Obiekt (taki jak "myCar") nie ma jednak wartości domyślnej. Jeśli zapomnisz zainicjować obiekt, przetwarzanie da mu wartość zerową. null nic nie znaczy. Nie zero. Nie negatywny. Całkowicie nicość. Pustka. Jeśli napotkasz błąd w wiadomości okno z napisem "NullPointerException" (i jest to dość powszechny błąd), ten błąd jest najprawdopodobniej spowodowany przez zapomnienie o inicjalizacji obiektu.

Krok 3. Używanie obiektu

Po pomyślnym zadeklarowaniu i zainicjowaniu zmiennej obiektowej możemy jej użyć. Używanie obiektu wymaga wywoływania funkcji wbudowanych w ten obiekt. Obiekt ludzki może jeść, samochód

może jeździć, pies może szczekać. Funkcje znajdujące się w obiekcie są technicznie określane jako "metody" w Javie, więc możemy zacząć używać tej nomenklatury. Wywołanie metody wewnątrz obiektu odbywa się za pomocą składni kropkowej: variableName.objectMethod (Argumenty metod); W przypadku samochodu żadna z dostępnych funkcji nie ma argumentu, więc wygląda tak:

myCar.draw ();

myCar.display ();

Ćwiczenie 8-3: Załóżmy, że istnieje klasa Ludzka. Chcesz napisać kod, aby zadeklarować obiekt ludzki, a także wywołać funkcję sleep () na tym obiekcie. Napisz poniższy kod:

Zadeklaruj i zainicjuj obiekt Ludzki: _____

Wywołaj funkcję sleep (): _____

8.5 Łączenie z kartą

Teraz, kiedy nauczyliśmy się definiować klasę i używamy obiektu zrodzonego z tej klasy, możemy pobrać kod z sekcji 8.2 i 8.3 i złożyć je razem w jednym programie.

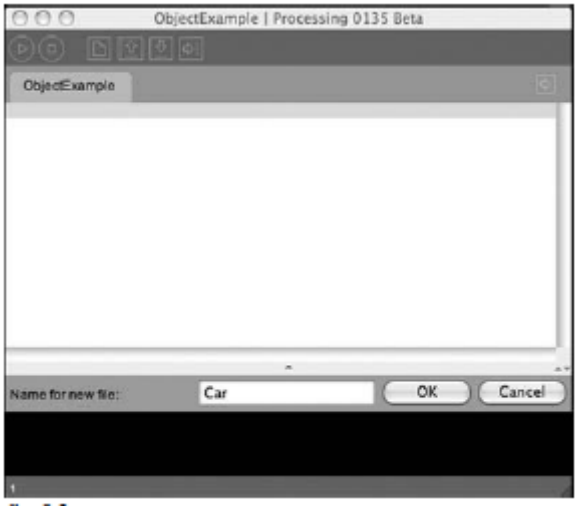
Przykład 8-1: Klasa samochodu i obiekt samochodu

```
Car myCar;  // Zadeklaruj obiekt samochodu jako zmienną globlną

void setup() {

size(200,200);

// Initialize Car object

myCar = new Car(); // Inicjalizuj obiekt samochodu w setup (), wywołując konstruktora.

}

void draw() {

background(0);

// Operate Car object.

myCar.move(); //Obsługuj obiekt samochodu w funkcji draw (), wywołując metody obiektów za
                //pomocą składni kropek

myCar.display();

}

class Car { // Zdefiniuj klasę poniżej reszty programu

color c;

float xpos;  // Zmienne

float ypos;

float xspeed;
```

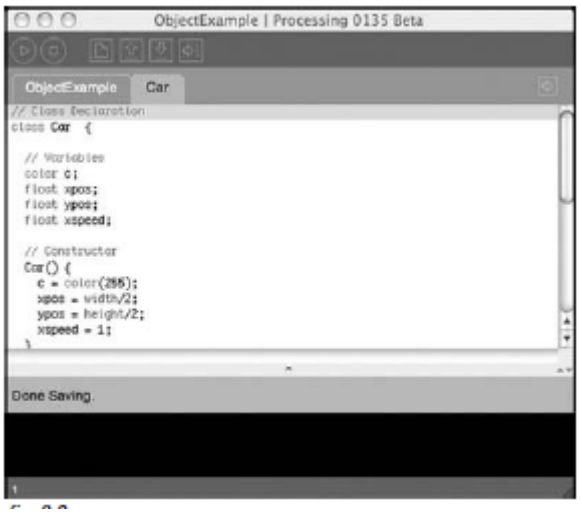
```
Car() {   // Konstruktor
c = color(255);
xpos = width/2;
ypos = height/2;
xspeed = 1;
}
void display() {    // Funkcja
// The car is just a square
rectMode(CENTER);
fill(c);
rect(xpos,ypos,20,10);
}
void move() {   // Funkcja
xpos = xpos + xspeed;
if (xpos > width) {
xpos = 0;
}
}
}
```

Zauważysz, że blok kodu, który zawiera klasę samochodów, znajduje się poniżej głównej części programu (pod rysunkiem ()). Miejsce to jest identyczne z miejscem, w którym umieściliśmy funkcje zdefiniowane przez użytkownika w części 7. Technicznie rzecz biorąc, kolejność nie ma znaczenia, o ile bloki kodu (zawarte w nawiasach klamrowych) pozostają nienaruszone. Klasa Car może przejść powyżej setup () lub może nawet przejść pomiędzy setup () i draw (). Każde miejsce jest technicznie poprawne, programując, fajnie jest umieszczać rzeczy, które wydają się najbardziej logiczne naszemu ludzkiemu mózgowi, a dolna część kodu jest dobrym punktem wyjścia. Niemniej jednak, Przetwarzanie wyłącza użyteczne środki do oddzielania bloków kodu od siebie za pomocą kart. W oknie przetwarzania znajdź strzałkę wewnątrz kwadratu w prawym górnym rogu. Jeśli klikniesz ten przycisk, zobaczysz, że wyłącza on opcję "Nowa karta" pokazaną na rysunku

Po wybraniu "Nowa karta" zostaniesz poproszony o wpisanie nazwy nowej zakładki, jak pokazano na rysunku



Chociaż możesz wybrać dowolne imię, prawdopodobnie dobrym pomysłem jest nazwanie karty po zajęciach, które zamierzasz tam umieścić. Następnie możesz wpisać główną treść kodu na jednej karcie (zatytułowanej "objectExample" na rysunku powyżej) i wpisać kod dla swojej klasy w innym (zatytułowanym "Samochód"). Przełączanie pomiędzy kartami jest proste, wystarczy kliknąć nazwę samej karty, jak pokazano na rysunku



Należy również zauważyć, że po utworzeniu nowej karty nowy plik .pde zostanie utworzony w folderze szkicu, jak pokazano na rysunku

Program zawiera zarówno plik ObjectExample.pde, jak i plik Car.pde.

Ćwiczenie 8-4: Utwórz szkic z wieloma zakładkami. Postaraj się, aby przykład Car'a działał bez błędów.

8.6 Argumenty konstruktora

W poprzednich przykładach obiekt samochodu został zainicjowany za pomocą nowego operatora, a następnie konstruktora klasy.

Car myCar = new Car ();

To było przydatne uproszczenie, a my nauczyliśmy się podstaw OOP. Niemniej jednak istnieje poważny problem z powyższym kodem. Co by było, gdybyśmy chcieli napisać program z dwoma przedmiotami samochodowymi?

// Tworzenie dwóch obiektów samochodowych

Car myCar1 = new Car ();

Car myCar2 = new Car ();

Osiąga nasz cel; kod wygeneruje dwa obiekty samochodowe, jeden przechowywany w zmiennej myCar1 i jeden w myCar2. Jednakże, jeśli uczysz się klasy Car, zauważysz, że te dwa samochody będą

identyczne: każdy będzie miał kolor biały, zacznie się na środku ekranu i będzie miał prędkość 1. W języku angielskim powyższe brzmi:

Zrób nowy samochód.

Chcemy zamiast tego powiedzieć:

Zrób nowy czerwony samochód, w miejscu (0,10) z prędkością 1.

Abyśmy mogli powiedzieć:

Zrób nowy niebieski samochód, w miejscu (0,100) z prędkością 2.

Możemy to zrobić, umieszczając argumenty wewnątrz metody konstruktora.

Car myCar = new Car (kolor (255,0,0), 0,100,2);

Konstruktor musi zostać przepisany, aby uwzględnić następujące argumenty:

Samochód (temp. Kolorów, zmiennoprzecinkowe, zmiennoprzecinkowe, zmiennoprzecinkowe) {

c = tempC;

xpos = tempXpos;

ypos = tempYpos;

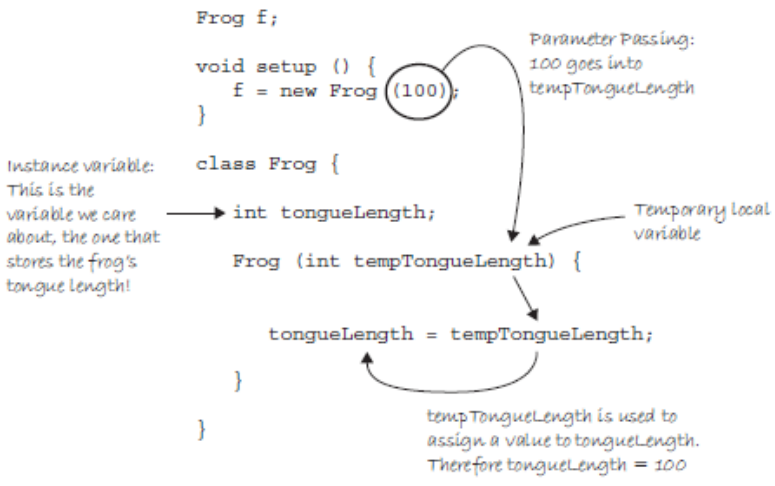
xspeed = tempXspeed;

}

Z mojego doświadczenia wynika, że użycie argumentów konstruktora do inicjowania zmiennych obiektu może być nieco oszałamiające. Proszę, nie obwiniaj się. Kod jest dziwnie wyglądający i może wydawać się bardzo niepotrzebny:

"Dla każdej zmiennej, którą chcę zainicjować w konstruktorze, muszę skopiować ją z tymczasowym argumentem do tego konstruktora? "

Niemniej jest to dość ważna umiejętność do nauki, a ostatecznie jest jedną z rzeczy, które sprawiają, że programowanie obiektowe jest potężne. Ale na razie może wydawać się bolesne. Przeanalizujmy ponownie parametr przekazujący parametry, aby zrozumieć, jak działa w tym kontekście. Patrz rysunek



Argumenty są zmiennymi lokalnymi używanymi w ciele funkcji, która jest wypełniana wartościami podczas wywoływania funkcji. W przykładach mają one tylko jeden cel, aby zainicjować zmienne wewnątrz obiektu. Są to zmienne, które się liczą, rzeczywisty samochód samochodu, jego aktualna pozycja x, i tak dalej. Argumenty konstruktora są tylko tymczasowe i istnieją tylko po to, aby przekazać wartość z miejsca, w którym obiekt jest tworzony do samego obiektu. To pozwala nam tworzyć różne obiekty przy użyciu tego samego konstruktora. Możesz również po prostu wpisać słowo "temp" w nazwie argumentu, aby przypomnieć o tym, co się dzieje (c vs. tempC). Zobaczycie również, że programiści używają podkreślenia (c vs. c_) w wielu przykładach. Możesz oczywiście nazwać te, co tylko chcesz. Jednak wskazane jest, aby wybrać nazwę, która ma dla ciebie sens, a także zachować spójność. Możemy teraz spojrzeć na ten sam program z wieloma instancjami obiektów, z których każda ma unikalne właściwości.

Przykład 8-2: Dwa obiekty Car

```
Car myCar1;

Car myCar2;  // Dwa obiekty

void setup() {

size(200,200);

myCar1 = new Car(color(255,0,0),0,100,2);

myCar2 = new Car(color(0,0,255),0,10,1);

}

void draw() {  // Parametry przechodzą do nawiasów, gdy obiekt jest konstruowany.

background(255);

myCar1.move();

myCar1.display();       // Mimo że istnieje wiele obiektów, wciąż potrzebujemy tylko jednej klasy. Bez
                        //względu na to, ile ciasteczek tworzymy, potrzebujemy tylko jednego
                        //obcinacza ciastek. Czy nie zmienia się programowanie obiektowe?

myCar2.move();

myCar2.display();

}

class Car {

color c;

float xpos;

float ypos;

float xspeed;

Car(color tempC, float tempXpos, float tempYpos, float tempXspeed) {

c = tempC;        // Konstruktor jest zdefiniowany z argumentami.
```

```
xpos = tempXpos;
ypos = tempYpos;
xspeed = tempXspeed;
}
void display() {
stroke(0);
fill(c);
rectMode(CENTER);
rect(xpos,ypos,20,10);
}
void move() {
xpos = xpos + xspeed;
if (xpos > width) {
xpos = 0;
}
}
}
```

Ćwiczenie 8-5: Przepisz przykład grawitacji z rozdziału 5, używając obiektów z klasą Ball. Uwzględnij dwie instancje obiektu Ball. Oryginalny przykład znajduje się tutaj, aby uzyskać odniesienie w ramach, które pomogą Ci zacząć.

```
_____ _______;
Ball ball2;
float grav = 0.1;
void setup() {
size(200,200);
ball1 = new _______(50,0,16);
__________________(100,50,32);
}
void draw() {
background(100);
ball1.display();
_____________________________
```

_____

_____

}

_____ {

float x;

_____

float speed;

float w;

_____(_____,_____,_____) {

x = _____;

_____

_____

speed = 0;

}

void _____() {

_____

_____

_____

}

_____

_____

_____

_____

_____

_____

}

Oryginał

// Simple gravity

float x = 100; // x

location

float y = 0; // y

```
location

float speed = 0; // speed

float gravity = 0.1;// gravity

void setup() {

size(200,200);

}

void draw() {

background(100);

// display the square

fill(255);

noStroke();

rectMode(CENTER);

rect(x,y,10,10);

// Add speed to y location

y = y + speed;

// Add gravity to speed

speed = speed + gravity;

// If square reaches the bottom

// Reverse speed

if (y > height) {

speed = speed * -0.95;

}

}
```

8.7 Obiekty są również typami danych!

Jest to nasze pierwsze doświadczenie z programowaniem obiektowym, dlatego chcemy to ułatwić. Przykłady w tej części wykorzystują tylko jedną klasę i składają najwyżej dwa lub trzy obiekty z tej klasy. Niemniej jednak, nie ma faktycznych ograniczeń. Szkic Processing może zawierać tyle klas, ile masz ochotę pisać. Jeśli np. Programowałeś grę Space Invaders, możesz utworzyć klasę Spaceship, klasę Enemy i klasę Bullet, używając obiektu dla każdej jednostki w grze. Ponadto, mimo że nie są prymitywne, klasy są typami danych, podobnie jak liczby całkowite i klatki. A ponieważ klasy składają się z danych, obiekt może zawierać inne obiekty! Na przykład załóżmy, że właśnie zakończyłeś programowanie klasy Widelec i Łyżka. Przechodząc do klasy klasy PlaceSetting, prawdopodobnie uwzględnisz zmienne dla obiektu Fork i obiektu Spoon wewnątrz samej klasy. Jest to całkowicie uzasadnione i dość powszechne w programowaniu obiektowym.

```
class PlaceSetting {

Fork fork;  // Klasa może zawierać inne obiekty wśród jej zmiennych.

Spoon spoon;

PlaceSetting() {

fork = new Fork();

spoon = new Spoon();

}

}
```

Obiekty, podobnie jak każdy typ danych, mogą być również przekazywane jako argumenty funkcji. Na przykładzie gry Space Invaders, jeśli statek kosmiczny wystrzeliłby pocisk na wroga, prawdopodobnie chcielibyśmy napisać funkcję wewnątrz klasy Wroga, aby ustalić, czy Wróg został trafiony przez pocisk.

```
void hit(Bullet b) {  // funkcja może mieć obiekt jako argument

// Code to determine if

// the bullet struck the enemy

}
```

W części 7 pokazaliśmy, w jaki sposób, gdy w funkcji przekazywana jest prymitywna wartość (całkowita, zmiennoprzecinkowa itp.), Tworzona jest kopia. W przypadku obiektów nie ma to miejsca, a wynik jest nieco bardziej intuicyjny. Jeśli wprowadzono zmiany do obiektu po przekazaniu go do funkcji, zmiany te będą miały wpływ na ten obiekt używany nigdzie indziej w całym szkicu. Th jest znane jako przekazywanie przez odniesienie, ponieważ zamiast kopii odwołanie do samego obiektu jest przekazywane do funkcji. Idąc dalej za pośrednictwem tej książki, a nasze przykłady stają się bardziej zaawansowane, zaczniemy aby zobaczyć przykłady, które korzystają z wielu obiektów, przechodzą obiektów do funkcji i więcej. W następnym rozdziale skupiono się na tworzeniu list obiektów. Część 10 omawia rozwój projektu obejmującego wiele klas. Na razie, gdy zamykamy rozdział ze Stworkiem będziemy trzymać się tylko jednej klasy.

8.8 Stworek zorientowany na obiekt

Niezmiennie pojawia się pytanie: "Kiedy powinienem używać programowania obiektowego? "Dla mnie odpowiedź jest zawsze. Obiekty pozwalają uporządkować koncepcje wewnątrz aplikacji na modułowe pakiety wielokrotnego użytku. Zobaczysz to wielokrotnie w trakcie tej książki. Jednak nie zawsze jest to wygodne i konieczne, aby rozpocząć każdy projekt przy użyciu orientacji obiektowej, szczególnie podczas nauki. Przetwarzanie ułatwia szybkie "szkicowanie" pomysłów wizualnych za pomocą kodu nieobiektywizowanego. W przypadku każdego projektu przetwarzania, który chcesz wprowadzić, moja rada to podejście krok po kroku. Nie musisz zaczynać pisania zajęć dla wszystkiego, co chcesz spróbować. Naszkicuj swój pomysł, pisząc kod w setup() i draw(). Zanurz się w logice tego, co chcesz robić i jak chcesz wyglądać. Kiedy twój projekt zacznie się rozwijać, poświęć trochę czasu, aby zreorganizować swój kod, być może najpierw z funkcjami, a następnie z obiektami. Całkowicie można poświęcić znaczną część swojego czasu na ten proces reorganizacji (często nazywany refaktoryzacją) bez dokonywania jakichkolwiek zmian w wyniku końcowym, czyli w tym, jak wygląda szkic i czy działa na ekranie. Dokładnie to robiliśmy z kosmonautą Zoogiem z Części 1 do teraz. Naszkicowaliśmy wygląd Stworka i eksperymentowaliśmy z niektórymi zachowaniami ruchu. Teraz, kiedy coś mamy, możemy

poświęcić czas na refaktor, czyniąc Zooga obiektem. Proces ten da nam siłę w programowaniu przyszłego życia Zooga w bardziej złożonych szkicach. Czas więc zejść i stworzyć klasę Zooga. Nasz mały Stworek jest prawie cały dorosły. Poniższy przykład jest praktycznie identyczny jak w przykładzie 7-5 (Stworek z funkcjami) z jedną zasadniczą różnicą. Wszystkie zmienne i wszystkie funkcje z przykładu 7-5 są teraz włączone do klasy Zoog za pomocą setup () i draw (), które zawierają prawie dowolny kod.

Przykład 8-3

```
Zoog zoog;  // Stworek jest obiektem!

void setup() {

size(200,200);

smooth();

zoog = new Zoog(100,125,60,60,16);     // Stworek otrzymuje początkowe właściwości za
                                       //pośrednictwem konstruktora

}

void draw() {

background(255);

// mouseX position determines speed factor

float factor = constrain(mouseX/10,0,5);

zoog.jiggle(factor);

zoog.display();  // Stworek może robić rzeczy z funkcjami!

}

class Zoog {

// Zoog's variables

float x,y,w,h,eyeSize;

// Zoog constructor

Zoog(float tempX, float tempY, float tempW, float tempH, float tempEyeSize) {

x = tempX;

y = tempY;        // Wszystko na temat Stworka zawarte jest w tej jednej klasie. Stworek ma właściwości
                  //lokalizacja, wysokość, rozmiar oczu), a Stworek ma zdolności (poruszanie się,
                  //wyświetlanie).

w = tempW;

h = tempH;

eyeSize = tempEyeSize;

}

// Move Zoog
```

```
void jiggle(float speed) {
// Change the location of Zoog randomly
x = x + random(-1,1)*speed;
y = y + random(-1,1)*speed;
// Constrain Zoog to window
x = constrain(x,0,width);
y = constrain(y,0,height);
}
// Display Zoog
void display() {
// Set ellipses and rects to CENTER mode
ellipseMode(CENTER);
rectMode(CENTER);
// Draw Zoog's arms with a for loop
for (float i = y - h/3; i < y + h/2; i + = 10) {
stroke(0);
line(x-w/4,i,x + w/4,i);
}
// Draw Zoog's body
stroke(0);
fill(175);
rect(x,y,w/6,h);
// Draw Zoog's head
stroke(0);
fill(255);
ellipse(x,y-h,w,h);
// Draw Zoog's eyes
fill(0);
ellipse(x-w/3,y-h,eyeSize,eyeSize*2);
ellipse(x + w/3,y – h,eyeSize,eyeSize*2);
// Draw Zoog's legs
```

```
stroke(0);

line(x – w/12,y + h/2,x – w/4,y + h/2 + 10);

line(x + w/12,y + h/2,x + w/4,y + h/2 + 10);

}

}
```

Ćwiczenie 8-6: Przepisz Przykład 8-3, aby dołączyć dwa Stworki. Czy możesz zmienić ich wygląd? Zachowanie? Rozważ dodanie koloru jako zmiennej Zoog.

Projekt Lekcji Trzeciej

Krok 1. Weź Projekt Lekcji Drugiej i przeorganizuj kod za pomocą funkcji.

Krok 2. Dokonaj reorganizacji kodu krok dalej, używając zmiennej klasy i obiektu.

Krok 3. Dodaj argumenty do Konstruktora klasy i spróbuj zrobić dwa lub trzy obiekty z różnymi zmiennymi.

Użyj przestrzeni podanej poniżej, aby szkicować projekty, notatki i pseudokod dla projektu.

IX Tablice

9.1 Tablice, dlaczego nam zależy?

Poświęćmy chwilę na ponowne zapoznanie się z przykładem samochodu z poprzedniego rozdziału na temat programowania obiektowego. Być może pamiętasz, że spędziliśmy dużo czasu na opracowaniu programu, który zawierał wiele instancji klasy, czyli dwa obiekty.

Car myCar1;

Car myCar2;

To naprawdę ekscytujący moment w rozwoju naszego życia jako programistów komputerowych. Prawdopodobnie zastanawiasz się nad nieco oczywistym pytaniem. Jak mogę to zrobić dalej i napisać

program z 100 przedmiotami samochodowymi? Dzięki sprytnemu kopiowaniu i wklejaniu możesz napisać program z następującym początkiem:

Car myCar1f

Car myCar2

Car myCar3

Car myCar4

Car myCar5

Car myCar6

Car myCar7

Car myCar8

Car myCar9

Car myCar10

Car myCar11

Car myCar12

Car myCar13

Car myCar14

Car myCar15

Car myCar16

Car myCar17

Car myCar18

Car myCar19

Car myCar20

Car myCar21

Car myCar22

Car myCar23

Car myCar24

Car myCar25

Car myCar26

Car myCar27

Car myCar28

Car myCar29

Car myCar30

Car myCar31

Car myCar32

Car myCar33

Car myCar34

Car myCar35

Car myCar36

Car myCar37

Car myCar38

Car myCar39

Car myCar40

Car myCar41

Car myCar42

Car myCar43

Car myCar44

Car myCar45

Car myCar46

Car myCar47

Car myCar48

Car myCar49

Car myCar50

Car myCar51

Car myCar52

Car myCar53

Car myCar54

Car myCar55

Car myCar56

Car myCar57

Car myCar58

Car myCar59

Car myCar60

Car myCar61

Car myCar62

Car myCar63

Car myCar64

Car myCar65

Car myCar66

Car myCar67

Car myCar68

Car myCar69

Car myCar70

Car myCar71

Car myCar72

Car myCar73

Car myCar74

Car myCar75

Car myCar76

Car myCar77

Car myCar78

Car myCar79

Car myCar80

Car myCar81

Car myCar82

Car myCar83

Car myCar84

Car myCar85

Car myCar86

Car myCar87

Car myCar88

Car myCar89

Car myCar90

Car myCar91

Car myCar92

Car myCar93

Car myCar94

Car myCar95

Car myCar96

Car myCar97

Car myCar98

Car myCar99

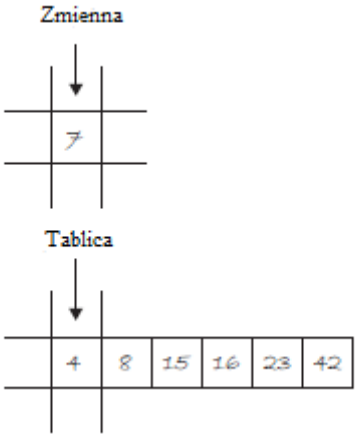
Car myCar100

Jeśli naprawdę chcesz sprawić sobie ból głowy, spróbuj wypełnić resztę programu wzorowanego na powyższym starcie. To nie będzie przyjemne przedsięwzięcie. Z pewnością nie zamierzam zostawić wam żadnego miejsca w książce, aby ćwiczyć. Tablica pozwoli nam pobrać te 100 linii kodu i umieścić je w jednej linii. Zamiast 100 zmiennych, tablica to jedna rzecz, która zawiera listę zmiennych. Za każdym razem, gdy program wymaga wielu wystąpień podobnych danych, może nadszedł czas, aby użyć tablicy. Na przykład tablica może służyć do przechowywania wyników czterech graczy w grze, wyboru 10 kolorów w programie do projektowania lub listy obiektów w symulacji akwarium

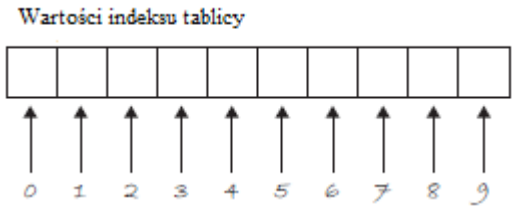
Ćwiczenie 9-1: Patrząc na wszystkie szkice, które do tej pory stworzyłeś, czy zasługujesz na użycie tablicy? Czemu?

9.2 Czym jest tablica?

Z części 4 można sobie przypomnieć, że zmienna to nazwany wskaźnik do miejsca w pamięci, w którym przechowywane są dane. Innymi słowy, zmienne pozwalają programom na śledzenie informacji przez pewien okres czasu. Tablica jest dokładnie taka sama, tylko zamiast wskazywać na pojedynczą informację, tablica wskazuje na wiele elementów.



Możesz myśleć o tablicy jako liście zmiennych. Należy zauważyć, że lista jest przydatna z dwóch ważnych powodów. Po pierwsze, lista śledzi same elementy na liście. Po drugie, lista śledzi kolejność tych elementów (który element jest pierwszy na liście, drugi, trzeci itp.). Jest to kluczowy punkt, ponieważ w wielu programach kolejność informacji jest tak samo ważna jak w przypadku sama informacja. W tablicy każdy element listy ma unikalny indeks, liczbę całkowitą, która określa jego pozycję na liście (element # 1, element # 2 itd.). We wszystkich przypadkach nazwa tablicy odnosi się do listy jako całości, podczas gdy każdy element jest dostępny poprzez jego pozycję. Zauważ, jak na rysunku, indeksy mieszczą się w zakresie od 0 do 9. Tablica ma w sumie 10 elementów, ale pierwszy numer elementu to 0, a ostatni element to 9. Możemy być kuszeni, by tupać nogami i narzekać: "Hej , dlaczego elementy nie są ponumerowane od 1 do 10? Czy to nie byłoby łatwiejsze? "



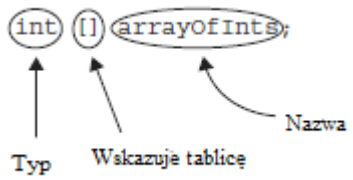
Początkowo intuicyjnie mogłoby się wydawać, że powinniśmy zaczynać od zera (a niektóre języki programowania), zaczynamy od zera, ponieważ technicznie pierwszy element tablicy znajduje się na początku tablicy, odległość zerowa od początku. Numerowanie elementów rozpoczynających się od 0 powoduje również, że wiele operacji na tablicach (proces wykonywania linii kodu dla każdego elementu listy) jest znacznie wygodniejsze. Kontynuując kilka przykładów, zaczniesz wierzyć w moc liczenia od zera.

Ćwiczenie 9-2: Jeśli macie tablicę zawierającą 1000 elementów, jaki jest zakres wartości indeksu dla tej tablicy?

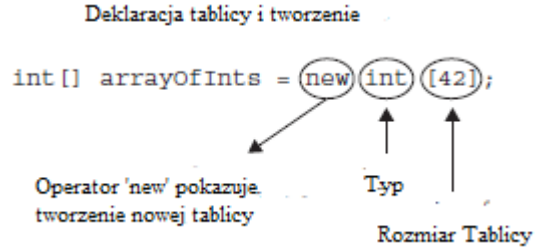
Odpowiedź: _____ do _____

9.3 Deklarowanie i tworzenie tablicy

W części 4 dowiedzieliśmy się, że wszystkie zmienne muszą mieć nazwę i typ danych. Tablice nie są różne. Deklaracja oświadczenia jednak wygląda inaczej. Oznaczamy użycie tablicy przez umieszczenie pustych nawiasów kwadratowych ("[]") po deklaracji typu. Zacznijmy od tablicy wartości pierwotnych, na przykład liczb całkowitych. (Możemy mieć tablice dowolnego typu danych, a wkrótce zobaczymy, jak utworzyć tablicę obiektów).



Deklaracja na powyższym rysunku wskazuje, że "arrayOf Ints" będzie przechowywać listę liczb całkowitych. Nazwa tablicy "arrayOf Ints" może być absolutnie dowolna (tylko słowo "tablica" w tym miejscu ilustruje to, czego się uczymy). Jedną podstawową własnością tablic jest jednak to, że mają one stałą wielkość. Po zdefiniowaniu rozmiaru tablicy, nigdy się nie zmieni. Lista 10 liczb całkowitych nigdy nie może przejść do 11. Ale gdzie w powyższym kodzie jest rozmiar zdefiniowanej macierzy? Nie jest. Kod po prostu deklaruje tablicę; musimy również upewnić się, że tworzymy rzeczywistą instancję tablicy o określonym rozmiarze. Aby to zrobić, używamy nowego operatora, w podobny sposób, jak w wywołaniu konstruktora obiektu. W przypadku obiektu mówimy "Utwórz nowy samochód" lub "Zrób nowy Zoog. "W przypadku tablicy mówimy" Utwórz nową tablicę liczb całkowitych "lub" Utwórz nową tablicę obiektów Car "i tak dalej. Zobacz deklarację tablicy na rysunku poniżej



Deklaracja tablicowa na powyższym rysunku pozwala nam określić rozmiar tablicy: ile elementów chcemy przechowywać w tablicy (lub, technicznie, ile pamięci w komputerze prosimy o przechowywanie naszych ukochanych danych). Piszemy to stwierdzenie w następujący sposób: nowy operator, a następnie typ danych, a następnie rozmiar tablicy ujęty w nawiasy kwadratowe. Ten rozmiar musi być liczbą całkowitą. Może to być zakodowana liczba, zmienna (typu integer) lub wyrażenie, które zwraca wartość całkowitą (np. 2 + 2).

float [] score = new float [4]; // Lista 4 liczb zmiennoprzecinkowych

Human [] people = new Human [100]; // Lista 100 obiektów ludzkich

int num = 50;

Car [] cars = new Car [num]; // Używanie zmiennej do określania rozmiaru

Spaceship [] ships = new Spaceship [num * 2 + 3]; // Używanie wyrażenia do określania rozmiaru

Ćwiczenie 9-3: Napisz deklaracje deklaracji dla następujących tablic:

30 liczb całkowitych

100 liczb zmiennoprzecinkowych

56 obiektów Stworka

Ćwiczenie 9-4: Które z poniższych deklaracji macierzy są poprawne i które są nieprawidłowe (i dlaczego)?

int[] numbers = new int[10];

float[] numbers = new float[5 + 6];

int num = 5;

float[] numbers = new int[num];

float num = 5.2;

Car[] cars = new Car[num];

int num = (5 * 6)/2;

float[] numbers = new

float[num = 5];

int num = 5;

Stworek [] zoogs = new Stworek[num * 10];

Rzeczy patrzą w górę. Nie tylko z powodzeniem zadeklarowaliśmy istnienie tablicy, ale daliśmy jej rozmiar i alokowaliśmy fizyczną pamięć dla przechowywanych danych. Brakuje jednak dużego elementu: danych przechowywanych w samej tablicy!

9.4 Inicjowanie tablicy

Jednym ze sposobów wypełnienia tablicy jest ścisłe kodowanie wartości przechowywanych w każdym miejscu tablicy.

Przykład 9-2: Inicjowanie elementów tablicy jeden po drugim

int [] stuff = new int [3];

stuff [0] = 8; // Pierwszy element tablicy to 8

stuff [1] = 3; // Drugi element tablicy to 3

stuff [2] = 1; // Trzeci element tablicy to 1

Jak widać, odwołujemy się do każdego elementu tablicy indywidualnie, określając indeks rozpoczynający się od 0. Składnia tego jest nazwą tablicy, a następnie wartością indeksu ujętą w nawiasy.

arrayName [INDEX]

Drugą opcją inicjowania tablicy jest ręczne wpisanie listy wartości ujętych w nawiasy klamrowe i oddzielonych przecinkami.

Przykład 9-3: Inicjowanie elementów tablicy naraz

int [] arrayOfInts = {1, 5, 8, 9, 4, 5};

float [] floatArray = {1,2, 3,5, 2,0, 3,4123, 9,9};

Ćwiczenie 9-5: Zadeklaruj tablicę trzech obiektów Stworka. Zainicjuj każdy punkt w tablicy za pomocą obiektu Stworka za pośrednictwem jego indeksu.

Stworek__ zoogs = nowy _____ [_____];

_____ [_____] = _____ _____ (100, 100, 50, 60, 16);

_____ [_____] = _____ _____ (_____);

_____ [_____] = _____ _____ (_____);

Oba te podejścia nie są powszechnie stosowane i nie zobaczysz ich w większości przykładów w całej książce. W rzeczywistości żadna metoda inicjalizacji naprawdę nie rozwiązała problemu postawionego na początku rozdziału. Wyobraź sobie inicjowanie każdego elementu pojedynczo z listą 100 lub (wdech) 1 000 lub (sapać z gazem!) 1 000 000 elementów. Rozwiązanie wszystkich naszych niepowodzeń obejmuje środki do iteracji przez elementy tablicy. Ding ding ding. Mam nadzieję, że w twojej głowie dzwoni głośny dzwonek. Pętle!

9.5 Operacje na macierzy

Zastanówmy się przez chwilę nad następującym problemem:

(A) Utwórz tablicę 1000 numerów punktów flotacyjnych. (B) Inicjalizuj każdy element tej tablicy z losową liczbą od 0 do 10.

Część A już wiemy, jak to zrobić.

float[] values = new float[1000];

To, czego chcemy uniknąć, to zrobić w części B:

values[0] = random(0,10);

values[1] = random(0,10);

values[2] = random(0,10);

values[3] = random(0,10);

values[4] = random(0,10);

values[5] = random(0,10);

etc. etc.

Opiszmy po angielsku, co chcemy zaprogramować: Dla każdej liczby n od 0 do 99, zainicjuj n element przechowywany w tablicy jako wartość losową pomiędzy 0 a 10. Przekształcając w kod, mamy:

```
int n = 0;

values[n] = random(0,10);

values[n + 1] = random(0,10);

values[n + 2] = random(0,10);

values[n + 3] = random(0,10);

values[n + 4] = random(0,10);

values[n + 5] = random(0,10);
```

Niestety sytuacja nie uległa poprawie. Mimo to zrobiliśmy duży krok naprzód. Używając zmiennej (n) do opisania indeksu w tablicy, możemy teraz użyć pętli while, aby zainicjować co n-ty element.

Przykład 9-4: Używanie pętli while do inicjowania wszystkich elementów tablicy

```
int n = 0;

while (n < 1000) {

values[n] = random(0,10);

n = n + 1;

}
```

Pętla for pozwala nam być jeszcze bardziej zwięzłym, jak pokazuje przykład 9-5.

```
for (int n = 0; n < 1000; n + + ) {

values[n] = random(0,10);

}
```

To, co kiedyś było 1000 linii kodu, to teraz trzy! Możemy wykorzystać tę samą technikę dla dowolnego typu operacji macierzy, którą chcielibyśmy wykonywać poza zwykłą inicjalizacją elementów. Na przykład możemy wziąć tablicę i podwoić wartość każdego elementu (użyjemy teraz od teraz zamiast n, ponieważ jest on częściej używany przez programistów).

Przykład 9-6: Operacja macierzy

```
for (int i = 0; i <1000; i + +) {

wartości [i] = wartości [i] * 2;

}
```

Jest jeden problem z Przykładem 9-6: użycie zakodowanej wartości 1000. Starając się być lepszymi programistami, zawsze powinniśmy kwestionować istnienie twardego kodu. W takim przypadku, co jeśli

chcieliśmy zmienić tablicę na 2000 elementów? Jeśli nasz program był bardzo długi z wieloma operacjami na macierzach, musielibyśmy wprowadzić tę zmianę wszędzie w całym kodzie. Na szczęście dla nas, rocessing daje nam dobry sposób na dynamiczne uzyskanie dostępu do rozmiaru tablicy, używając składni punktowej, której nauczyliśmy się dla obiektów w części 8. length jest właściwością każdej tablicy i możemy uzyskać do niej dostęp, mówiąc:

arrayName długość kropki Wykorzystajmy długość podczas usuwania tablicy. Th będzie obejmować resetowanie każdej wartości do 0.

Przykład 9-7: Operacja macierzy wykorzystująca długość kropki

```
for (int i = 0; i < values.length; i + + ) {

values[i] = 0;

}
```

Ćwiczenie 9-6: Zakładając tablicę 10 liczb całkowitych, czyli

```
int [] nums = {5,4,2,7,6,8,5,2,4,14};
```

napisz kod, aby wykonać następujące operacje tablicowe (Zwróć uwagę, że liczba wskazówek jest różna, tylko dlatego, że słowo [____] nie jest jawnie zapisane, nie oznacza, że nie powinno być nawiasów).

Podnieś do kwadratu każdą liczbę (tj. pomnóż każdą przez nią samą) :

```
for (int i ___; i < ____; i + + ) {

___[i] = _____*_____;

 }
```

Dodaj liczbę losową od zera do 10 na każdą liczbę

```
_____________________________________

____  + = int(_________);

__
```

Dodaj do każdego numeru liczbę występującą w tablicy. Pomiń ostatnią wartość w tablicy.

```
for (int i = 0; i < ____; i + + ) {

_____ + = _______ [____];

}
```

Oblicz sumę wszystkich liczb.

```
____ _________ = ___;

for (int i = 0; i < nums.length; i + + )

{

_______ + = _________;

}
```

9.6 Prosty przykład tablicy: Wąż

Pozornie banalne zadanie, programowanie śladu za myszą, nie jest tak łatwe, jak mogłoby się początkowo wydawać. To rozwiązanie wymaga tablicy, która będzie służyć do przechowywania historii lokalizacji myszy. Użyjemy dwóch tablic, jedna do przechowywania poziomych lokalizacji myszy i jedna do pionu. Powiedzmy, arbitralnie, że chcemy zapisać 50 ostatnich lokalizacji myszy. Najpierw deklarujemy dwie tablice.

int [] xpos = new int [50];

int [] ypos = new int [50];

Po drugie, w setup () musimy zainicjować tablice. Ponieważ na początku programu nie było żadnego ruchu myszy, po prostu wypełnimy tablice z zerami.

dla (int i = 0; i <xpos.length; i + +) {

xpos [i] = 0;

ypos [i] = 0;

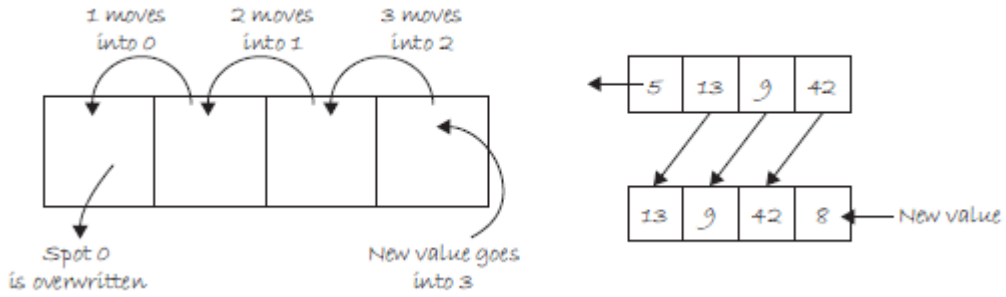
}

Za każdym razem, poprzez główną pętlę draw (), chcemy zaktualizować tablicę z bieżącą lokalizacją myszy. Zdecydujmy się umieścić aktualną lokalizację myszy w ostatnim miejscu tablicy. Długość tablicy wynosi 50, co oznacza, że wartości indeksu mieszczą się w przedziale 0-49. Ostatnie miejsce to indeks 49 lub długość tablicy minus jeden.

xpos[xpos.length–l] = mouseX; // Ostatnie miejsce w tablicy to długość minus jeden.

ypos[ypos.length–1] = mouseY;

Teraz jest trudna część. Chcemy zachować tylko ostatnie 50 lokalizacji myszy. Przechowując bieżącą lokalizację myszy na końcu tablicy, nadpisujemy to, co wcześniej było tam przechowywane. Jeśli mysz znajduje się w (10,10) podczas jednej ramki i (15,15) podczas drugiej, chcemy umieścić (10,10) w drugim od końca miejscu i (15,15) w ostatnim miejscu. Rozwiązaniem jest przesunięcie wszystkich elementów tablicy w dół o jeden punkt przed aktualizacją bieżącej lokalizacji. Jest to pokazane na rysunku



Indeks elementów 49 przenosi się do punktu 48, 48 przenosi się do punktu 47, 47 do 46 i tak dalej. Możemy to zrobić, przechodząc przez tablicę i ustawiając każdy indeks elementu i na wartość elementu i plus jeden. Uwaga: musimy zatrzymać się na drugiej do ostatniej wartości, ponieważ dla elementu 49 nie ma elementu 50 (49 plus 1). Innymi słowy, zamiast mieć warunek wyjścia

i < xpos.length;

zamiast tego musimy powiedzieć:

i < xpos.length - 1;

Pełny kod do wykonania tej zmiany tablicy jest następujący:

for (int i = 0; i <xpos.length-1; i + +) {

xpos [i] = xpos [i + 1];

ypos [i] = ypos [i + 1];

}

Wreszcie możemy użyć historii lokalizacji myszy, aby narysować serię okręgów. Dla każdego elementu tablicy xpos i tablicy ypos narysuj elipsę w odpowiednich wartościach przechowywanych w tablicy.

for (int i = 0; i <xpos.length; i + +) {

noStroke ();

fill (255);

ellipse (xpos [i], ypos [i], 32, 32);

}

Dzięki temu, że jest to nieco bardziej wyszukane, możemy połączyć jasność koła, a także wielkość koła z lokalizacją w tablicy, co oznacza, że wcześniejsze (a zatem i starsze) wartości będą jasne i małe, a później (nowsze) wartości będą ciemniejsze i większe. Osiąga się to za pomocą zmiennej zliczającej i, aby ocenić kolor i rozmiar.

for (int i = 0; i < xpos.length; i + +) {

noStroke();

fill(255 – i*5);

ellipse(xpos[i],ypos[i],i,i);

}

Łącząc cały kod, mamy następujący przykład, z danymi wyjściowymi pokazanymi na rysunku

Przykład 9-8: Wąż podążający za myszą

// x and y positions

int[] xpos = new int[50]; // Deklaruj dwie tablice z 50 elementami.

int[] ypos = new int[50];

```
void setup() {

size(200,200);

smooth();

// Initialize

for (int i = 0; i  < xpos.length; i++)

xpos[i] = 0;

ypos[i] = 0; // Zainicjuj wszystkie elementy każdej tablicy na zero.

}

}

void draw() {

background(255);

// Shift array values

for (int i = 0; i < xpos.length-1; i + + ) {

xpos [i] = xpos[i + 1];     //Przesuń wszystkie elementy w dół o jedno miejsce. xpos [0] = xpos [1], xpos
                            //[1] = xpos = [2], i tak dalej. Zatrzymaj się na drugim od ostatniego elementu.

ypos[i] = ypos[i + 1];

}

// New location

xpos[xpos.length–1] = mouseX;  // Zaktualizuj ostatnie miejsce w tablicy za pomocą lokalizacji myszy.

ypos[ypos.length–1] = mouseY;

// Draw everything

for (int i = 0; i < xpos.length; i + + ) {

noStroke();      //Narysuj elipsę dla każdego elementu w tablicach. Kolor i rozmiar są powiązane z
                 //licznikiem pętli: i.

fill(255-i*5);

ellipse(xpos[i],ypos[i],i,i);

}

}
```

Ćwiczenie 9-7: Przepisz przykład węża w sposób zorientowany obiektowo na klasę Snake. Czy potrafisz zrobić węże o nieco odmiennym wyglądzie (różne kształty, kolory, rozmiary)? (W przypadku zaawansowanego problemu utwórz klasę Point, która przechowuje współrzędne X i Y jako część szkicu. Każdy obiekt węższy będzie miał tablicę obiektów Point, zamiast dwóch oddzielnych tablic wartości xiy. Dotyczy to tablic obiektów, omówionych w następnej sekcji).

9.7 Tablice obiektów

Wiem wiem. Wciąż nie odpowiedziałem w pełni na to pytanie. Jak napisać program zawierający 100 obiektów samochodowych? Jedną z najładniejszych cech łączenia programowania obiektowego z tablicami jest prostota przenoszenia programu z jednego obiektu na 10 obiektów na 10 000 obiektów. W rzeczywistości, gdybyśmy byli ostrożni, nie będziemy musieli zmieniać klasy samochodu. Klasa nie dba o to, ile obiektów jest z niej zrobionych. Zakładając, że zachowamy identyczny kod klasy samochodu, spójrzmy, jak rozszerzamy program główny, aby użyć tablicy obiektów zamiast jednego. Powróćmy do głównego programu dla jednego obiektu Car.

```
Car myCar;

void setup() {

myCar = new Car(color(255,0,0),0,100,2);

}

void draw() {

background(255);

myCar.move();

myCar.display();

}
```

W powyższym kodzie są trzy kroki i musimy zmienić każdy z nich, aby uwzględnić tablicę.

PRZED	**PO**
Zadeklaruj samochód	Zadeklaruj Car Array
Car myCar;	Car[] cars = new Car[100];
Zainicjuj samochód	Zainicjuj każdy element tablicy samochodowej
myCar = new Car(color(255),0,100,2);	for (int i = 0; i < cars.length; i + +) {
	cars[i] = new Car(color(i*2),0,i*2,i);
	}
Uruchom samochód, wywołując metody	Uruchom każdy element tablicy samochodowej
myCar.move();	
myCar.display();	dla (int i = 0; i <cars.length; i + +) {
	cars [i] .move ();
	cars [i] .display ();
	}

Pozostawia nam to przykład 9-9. Zwróć uwagę, że zmiana liczby samochodów obecnych w programie wymaga jedynie zmiany definicji tablicy. Nic innego nie musi się zmieniać!

Przykład 9-9: Tablica obiektów samochodowych

```
Car[] cars = new Car[100] ; // Tablica 100 obiektów Car!

void setup() {

size(200,200);

smooth();

for (int i = 0; i < cars.length; i + + ) {

cars[i] = new Car(color(i*2),0,i*2,i/20.0); //Zainicjuj każdy samochód za pomocą pętli for.

}

}

void draw() {

background(255);  // Uruchom każdy samochód za pomocą pętli for.

for (int i = 0; i < cars.length; i + + ) {

cars[i].move();

cars[i].display();

}

}

class Car {         // Klasa Car nie zmienia się, czy ją robimy samochód, 100 samochodów lub
          //1000  samochodów!

color c;

float xpos;

float ypos;

float xspeed;

Car(color c_, float xpos_, float ypos_, float xspeed_) {

c = c_;

xpos = xpos_;

ypos = ypos_;

xspeed = xspeed_;

}

void display() {

rectMode(CENTER);
```

```
stroke(0);

fill(c);

rect(xpos,ypos,20,10);

}

void move() {

xpos = xpos + xspeed;

if (xpos > width) {

xpos = 0;

}

}

}
```

9.8 Obiekty interaktywne

Kiedy dowiedzieliśmy się o zmiennych (Partr 4) i warunkowych (Część 5), zaprogramowaliśmy prosty efekt rollover. Prostokąt pojawia się w oknie i jest jednym kolorem, gdy mysz znajduje się na górze, a inny kolor, gdy mysz nie jest. Poniżej znajduje się przykład, który przyjmuje ten prosty pomysł i umieszcza go w obiekcie "Stripe". Mimo że istnieje 10 pasków, każda z nich reaguje indywidualnie na mysz, posiadając własną funkcję rollover ().

```
void rollover(int mx, int my) {

if (mx > x & & mx < x + w) {

mouse = true;

} else {

mouse = false;

}

}
```

Ta funkcja sprawdza, czy punkt (mx, my) znajduje się w pionowym pasku. Czy jest większy niż lewa krawędź i mniejszy niż prawy brzeg? Jeśli tak, zmienna boolowska "mouse" jest ustawiona na true. Podczas projektowania klas często wygodnie jest używać zmiennej boolowskiej do śledzenia właściwości obiektu przypominającego przełącznik. Na przykład obiekt samochodu może być uruchomiony lub nie działać. Zoog może być szczęśliwy lub niezadowolony. Ta zmienna binarna jest używana w instrukcji warunkowej wewnątrz funkcji wyświetlania () obiektu Stripe w celu określenia koloru Paska.

```
void display() {

if (mouse) {

fill(255);

} else {
```

```
fill(255,100);

}

noStroke();

rect(x,0,w,height);

}
```

Kiedy wywołujemy funkcję rollover () na tym obiekcie, możemy następnie przekazać jako argumenty mouseX i mouseY.

```
stripes[i].rollover(mouseX,mouseY);
```

Nawet gdybyśmy mogli uzyskać dostęp do mouseX i mouseY bezpośrednio w funkcji rollover (), lepiej jest użyć argumentów. Th pozwala na większą elastyczność. Obiekt Stripe może sprawdzić i określić, czy dowolna współrzędna x, y znajduje się w jego prostokącie. Być może później, będziemy chcieli, aby pasek zmienił kolor na biały, gdy nad nim znajduje się inny obiekt, a nie mysz. Oto pełny przykład "pasków interaktywnych"

Przykład 9-10: Paski interaktywne

```
// An array of stripes

Stripe[] stripes = new Stripe[10];

void setup() {

size(200,200);

// Initialize all " stripes "

for (int i = 0; i < stripes.length; i + + ) {

stripes[i] = new Stripe();

}

}

void draw() {

background(100);  // Przekazywanie współrzędnych myszy do obiektu.

// Move and display all " stripes "

for (int i = 0; i < stripes.length; i + + ) {

// Check if mouse is over the Stripe

stripes[i].rollover(mouseX,mouseY);

stripes[i].move();

stripes[i].display();

}

}
```

```
class Stripe {
float x; // horizontal location of stripe
float speed; // speed of stripe
float w; // width of stripe
boolean mouse; // state of stripe (mouse is over or not?) // Zmienna boolowska śledzi stan obiektu.
Stripe() {
x = 0; // All stripes start at 0
speed = random(1); // All stripes have a random positive speed
w = random(10,30);
mouse = false;
}
// Draw stripe
void display() {
if (mouse) {  // Zmienna Boolean określa kolor Stripe.
fill(255);
} else {
fill(255,100);
}
noStroke();
rect(x,0,w,height);
}
// Move stripe
void move() {
x + = speed;
if (x > width + 20) x = – 20;
}
// Check if point is inside of Stripe
void rollover(int mx, int my) {  // Sprawdź, czy punkt (mx, my) znajduje się wewnątrz paska.
// Left edge is x, Right edge is x + w
if (mx > x & & mx < x + w) {
mouse = true;
```

```
} else {
mouse = false;
}
}
}
```

Ćwiczenie 9-8: Napisz klasę Button (patrz Przykład 5-5 f lub przycisk nieobiektowy). Klasa przycisku powinna się zarejestrować po naciśnięciu przycisku myszy i zmianie koloru. Utwórz obiekty przycisków o różnych rozmiarach i lokalizacjach za pomocą tablicy. Przed napisaniem programu głównego, wykreśl klasę Button. Załóżmy, że przycisk jest wyłączony, gdy pojawi się pierwszy raz. Oto fragment kodu:

```
class Button {
float x;
float y;
float w;
float h;
boolean on;
Button(float tempX, float tempY, float tempW, float tempH) {
x = tempX;
y = tempY;
w = tempW;
h = tempH;
on = _____;
}
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

}

9.9 Funkcje macierzy przetwarzania

OK, więc muszę się spowiadać. Kłamałem. Cóż, w pewnym sensie. Zobacz, wcześniej w tym rozdziale, bardzo podkreśliłem, że po ustawieniu rozmiaru tablicy, nigdy nie możesz zmienić tego rozmiaru. Kiedy już to zrobisz wykonałeś 10 obiektów Button, nie możesz zrobić jedenastego. I podtrzymuję te oświadczenia. Technicznie rzecz biorąc, przydzielając 10 miejsc w tablicy, powiedziałeś Processing dokładnie, ile miejsca w pamięci chcesz użyć. Nie można oczekiwać, że blok pamięci będzie miał więcej miejsca obok niego, aby można było powiększyć rozmiar tablicy. Nie ma jednak powodu, dla którego nie moglibyśmy po prostu utworzyć nowej tablicy (takiej, która zawiera 11 punktów), skopiować pierwsze 10 z oryginalnej tablicy i wrzucić nowy obiekt przycisku w ostatnim miejscu. Przetwarzanie w rzeczywistości oferuje zestaw funkcji tablicowych, które manipulują rozmiarem macierzy, zarządzając tym procesem. Są to: shorten (), concat (), subset (), append (), splice () i expand (). Ponadto istnieją funkcje do zmiany kolejności w tablicy, takie jak sort () i reverse (). Szczegóły dotyczące wszystkich tych funkcji można znaleźć w odnośniku. Spójrzmy na jeden przykład, który używa append () do rozszerzenia rozmiaru tablicy. Przykład Th (który zawiera odpowiedź na Ćwiczenie 8-5) zaczyna się od tablicy jednego obiektu. Za każdym naciśnięciem myszy tworzony jest nowy obiekt i dołączany do końca oryginalnej tablicy.

Przykład 9-11: Zmienianie rozmiaru tablicy za pomocą append ()

```
Ball[] balls = new Ball[1];  // Zaczynamy od tablicy z jednym elementem.

float gravity = 0.1;

void setup() {

size(200,200);

smooth();

frameRate(30);

// Initialize ball index 0

balls[0] = new Ball(50,0,16);

}

void draw() {

background(100);

// Update and display all balls

for (int i = 0; i < balls.length; i + + ) {      // Niezależnie od długości tej tablicy, zaktualizuj i wyświetl
                                                 //wszystkie obiekty.

balls[i].gravity();
```

```
balls[i].move();

balls[i].display();

}

}

void mousePressed() {

// A new ball object

Ball b = new Ball(mouseX,mouseY,10); //Utwórz nowy obiekt w lokalizacji myszy.

// Append to array

balls = (Ball[]) append(balls,b);  //Tutaj funkcja, append () dodaje element na końcu tablicy. append ()
                                   //pobiera dwa argumenty. Pierwsza to tablica, do której chcesz
                                   //dołączyć,a druga to ta, którą chcesz dodać. Musisz ponownie
                                   //przypisać wynik funkcji append () do oryginalnej tablicy. Ponadto
                                   //funkcja append() wymaga jawnego  określenia typu danych w
                                   //tablicy, umieszczając typ danych tablicy w nawiasach: "(Ball [])". To
                                   //się nazywa casting

}

class Ball {

float x;

float y;

float speed;

float w;

x = tempX;

y = tempY;

w = tempW;

speed = 0;

}

void gravity() {

// Add gravity to speed

speed = speed + gravity;

}

void move() {

// Add speed to y location

y = y + speed;
```

```
// If square reaches the bottom

// Reverse speed

if (y > height) {

speed = speed * –0.95;

y = height;

}

}

void display() {

// Display the circle

fill(255);

noStroke();

ellipse(x,y,w,w);

}

}
```

Innym sposobem posiadania tablicy skalowalnej jest użycie specjalnego obiektu znanego jako ArrayList.

9.10 Tysiąc i jeden Stworek

Nadszedł czas, aby zakończyć podróż Stworka i spojrzeć na to, jak przechodzimy od jednego obiektu Stworek do wielu. W ten sam sposób, w jaki wygenerowaliśmy tablicę Car lub tablicę Stripe, możemy po prostu skopiować dokładnie klasę Stworek utworzoną w przykładzie 8-3 i zaimplementować tablicę.

Przykład 9-12: 200 obiektów Stworka w tablicy

```
Zoog[] zoogies = new Zoog[200];

void setup() {

size(400,400);

smooth();

for (int i = 0; i < zoogies.length; i + + ) {

zoogies[i] = new Zoog(random(width),random(height),30,30,8);

// Jedyną różnicą między tym przykładem a poprzednim rozdziałem (przykład 8-3) jest użycie tablicy
//dla wielu obiektów Zooga.

}

}

void draw() {
```

```
background(255); // Draw a black background
for (int i = 0; i < zoogies.length; i + + ) {
zoogies[i].display();
zoogies[i].jiggle();
}
}
class Zoog {
// Zoog's variables
float x,y,w,h,eyeSize;
// Zoog constructor
Zoog(float tempX, float tempY, float tempW, float tempH, float tempEyeSize) {
x = tempX;
y = tempY;
w = tempW;
h = tempH;
eyeSize = tempEyeSize;
}
// Move Zoog
void jiggle() {    // Dla uproszczenia usunęliśmy również argument "speed" z funkcji jiggle (). Spróbuj
                   //dodać go ponownie jako ćwiczenie
// Change the location
x = x + random(–1,1);
y = y + random(–1,1);
// Constrain Zoog to window
x = constrain(x,0,width);
y = constrain(y,0,height);
}
// Display Zoog
void display() {
// Set ellipses and rects to CENTER mode
ellipseMode(CENTER);
```

```
rectMode(CENTER);
// Draw Zoog's arms with a for loop
for (float i = y-h/3; i < y + h/2; i + = 10) {
stroke(0);
line(x–w/4,i,x + w/4,i);
}
// Draw Zoog's body
stroke(0);
fill(175);
rect(x,y,w/6,h);
// Draw Zoog's head
stroke(0);
fill(255);
ellipse(x,y-h,w,h);
// Draw Zoog's eyes
fill(0);
ellipse(x–w/3,y–h,eyeSize,eyeSize*2);
ellipse(x + w/3,y–h,eyeSize,eyeSize*2);
// Draw Zoog's legs
stroke(0);
line(x–w/12,y + h/2,x-w/4,y + h/2 + 10);
line(x + w/12,y + h/2,x + w/4,y + h/2 + 10);
}
}
```

Projekt Lekcji Czwartej

Krok 1. Weź klasę, którą stworzyłeś w lekcji i przygotuj tablicę obiektów z tej klasy.

Krok 2. Czy obiekty mogą reagować na mysz? Spróbuj użyć funkcji dist (), aby określić bliskość obiektu do myszy. Na przykład, czy możesz sprawić, by każdy obiekt bardziej się poruszał, im bliżej myszy? Ile obiektów możesz zrobić, zanim szkic będzie działać zbyt wolno? Użyj przestrzeni podanej poniżej, aby szkicować projekty, notatki i pseudokod dla projektu.

X. Algorytmy

10.1 Gdzie byliśmy? Gdzie idziemy?

Nasz przyjaciel Stworek miał niezły bieg. Stworek nauczył nas podstaw bibliotek kształtu rysunku w przetwarzaniu. Stamtąd Stworek podchodził do interakcji z myszą, do samodzielnego poruszania się za pomocą zmiennych, zmiany kierunku za pomocą warunków, rozszerzania ciała za pomocą pętli, organizowania swojego kodu za pomocą funkcji, enkapsulacji jego danych i funkcjonalności do obiektu, i wreszcie powielania się za pomocą tablica. To dobra historia, która dobrze nas traktowała. Jest jednak bardzo mało prawdopodobne, aby wszystkie projekty programistyczne, które zamierzasz wykonać po przeczytaniu tej książki, będą zawierały kolekcję obcych istot poruszających się po ekranie (jeśli tak, to jesteś szczęśliwym programistą!). Musimy zatrzymać się na chwilę i rozważyć, czego się nauczyliśmy i jak można zastosować to, co chcesz robić. Jaki jest twój pomysł i jak mogą ci pomóc zmienne, warunki, pętle, funkcje, obiekty i tablice? We wcześniejszych częściach skupiliśmy się na prostych przykładach programowania "jednego elementu". Stworek poruszałby się i kręcił tylko. Stworek nie zaczął nagle podskakiwać. Stworek był zwykle sam, nigdy nie wchodząc w interakcję z innymi obcymi stworzeniami po drodze. Oczywiście moglibyśmy wziąć te wczesne przykłady dalej, ale wtedy ważne było, aby trzymać się podstawowych funkcji, tak abyśmy mogli naprawdę poznać podstawy. W świecie rzeczywistym projekty oprogramowania zwykle obejmują wiele ruchomych części. Rozdział ten ma na celu zademonstrowanie, w jaki sposób większy projekt powstaje z wielu mniejszych programów "jednofunkcyjnych", takich jak te, z którymi zaczynamy czuć się komfortowo. Ty, programista, zaczniesz od ogólnej wizji, ale musisz nauczyć się, jak podzielić ją na poszczególne części, aby pomyślnie zrealizować tę wizję. Zaczniemy od pomysłu. Najlepiej byłoby wybrać przykładowy "pomysł", który mógłby stanowić podstawę dla każdego projektu, który chcesz stworzyć po przeczytaniu tej książki. Niestety, nie ma czegoś takiego. Programowanie własnego oprogramowania jest niezwykle ekscytujące ze względu na niezmierzoną gamę możliwości tworzenia. Ostatecznie będziesz musiał stworzyć swoją własną drogę. Jednak, tak jak wybraliśmy prostą istotę do nauki podstaw, wiedząc, że tak naprawdę nie będziemy programować stworzeń przez całe nasze życie, możemy spróbować dokonać ogólnego wyboru, który, miejmy nadzieję, posłuży zdobyciu wiedzy na temat procesu opracowywania większych projektów. . Nasz wybór będzie prostą grą z interaktywnością, wieloma obiektami i celem. Skupiamy się nie na dobrym projekcie gry, ale na dobrym projektowaniu oprogramowania. Jak przejść od myśli do kodu? Jak zaimplementować własny algorytm, aby zrealizować swoje pomysły? Przekonamy się, jak duży projekt dzieli się na cztery mini-projekty i atakuje je jeden po drugim, ostatecznie doprowadzając wszystkie części do realizacji pierwotnego pomysłu. Będziemy nadal podkreślać programowanie obiektowe, a każda z tych części będzie rozwijana przy użyciu klasy. Korzyścią będzie przekonanie się, jak łatwo jest stworzyć końcowy program, łącząc samodzielne, w pełni funkcjonalne klasy. Zanim przejdziemy do pomysłu i jego części, przyjrzyjmy się koncepcji algorytmu, który będziemy potrzebować w krokach 2a i 2b.

<u>Nasz proces</u>

1. Pomysł - Rozpocznij od pomysłu.

2. Części - przerzuć pomysł na mniejsze części.

a. Algorytm Pseudokod: Dla każdej części opracuj algorytm dla tej części w pseudokod.

b . Kod algorytmu - wprowadź algorytm z kodem.

c. Obiekty - Pobierz dane i funkcje związane z tym algorytmem i zbuduj je w klasie.

3. Integracja - weź wszystkie klasy z kroku 2 i włącz je do jednego większego algorytmu

10.2 Algorytm: Tańcz do rytmu własnego bębna.

Algorytm to procedura lub formuła rozwiązania problemu. W programowaniu komputerowym algorytm to sekwencja kroków wymaganych do wykonania zadania. Każdy przykład, który do tej pory stworzyliśmy w tym tekście, dotyczył algorytmu. Algorytm nie jest zbyt odległy od przepisu.

1. Rozgrzej piekarnik do 200 °C.

2. Umieść cztery piersi z kurczaka bez kości w naczyniu do pieczenia.

3. Rozłóż musztardę równomiernie na kurczaku.

4. Piecz w temperaturze 200 °C przez 30 minut.

Powyższe jest dobrym algorytmem do gotowania kurczaka z gorczycy. Najwyraźniej nie zamierzamy pisać programu obróbki, aby gotować kurczaka. Niemniej jednak, jeśli byśmy to zrobili, powyższy pseudokod może zmienić się w następujący kod.

```
preheatOven(200);

placeChicken(4, " baking dish "167 );

spreadMustard();

bake(200,30);
```

Przykład, który wykorzystuje algorytm do rozwiązania problemu matematycznego, jest bardziej odpowiedni dla naszych zainteresowań. Opiszmy algorytm do oceny sumy sekwencji liczb od 1 do N.

SUMA (N) = 1 + 2 + 3 + ... + N

gdzie N jest dowolną liczbą całkowitą większą od zera.

1. Ustaw SUM = 0 i licznik I = 1

2. Powtórz następujące kroki, gdy ja jest mniejsze lub równe N.

za. Oblicz SUMA = I i zapisz wynik w SUMIE.

b. Zwiększ wartość I o 1.

3. Rozwiązanie jest teraz numerem zapisanym w SUM.

Przekładając powyższy algorytm na kod, mamy:

```
int sum = 0;

int n = 10;

int i = 0; // Krok 1. Ustaw sumę równą 0 i licznikowi i 0.

while (i <= n) {  // Krok 2. Powtórzyć, gdy i <= n.

sum = sum + i; // Krok 2a. Przyrost sumy.

i + +; // Krok 2b. Przyrost i.

}
```

print ln (suma); // Krok 3. Rozwiązanie jest w sumie. Wyświetl sumę!

Tradycyjnie programowanie jest uważane za proces (1) rozwijania pomysłu, (2) wypracowanie algorytmu do realizacji tego pomysłu i (3) wypisanie kodu do wdrożenia tego algorytmu. To właśnie osiągnęliśmy zarówno w przypadku kurczaka, jak i sumy. Niektóre pomysły są jednak zbyt duże, by można je było ukończyć za jednym zamachem. Dokonamy przeglądu tych trzech kroków i stwierdzimy, że programowanie jest procesem (1) opracowania idei, (2) podzielenia tego pomysłu na mniejsze części, które można zarządzać, (3) opracowania algorytmu dla każdej części, (4) pisania kod każdej części, (5) opracowanie algorytmu dla wszystkich części razem i (6) integracja kodu dla wszystkich części razem. Nie oznacza to, że nie należy eksperymentować po drodze, nawet całkowicie zmieniając pierwotny pomysł. I oczywiście, po zakończeniu kodu, prawie zawsze pozostanie praca do wykonania w zakresie czyszczenia kodu, ustawień błędów i dodatkowych funkcji. Jednak taki właśnie proces myślenia powinien poprowadzi Cię od pomysłu do kodu. Jeśli będziesz ćwiczył rozwój swoich projektów za pomocą tej strategii, tworzenie kodu, który implementuje twoje pomysły, może być mniej zniechęcające.

10.3 Od pomysłu do części

Aby przećwiczyć naszą strategię rozwoju, zaczniemy od pomysłu, bardzo prostej gry. Zanim zdążymy się zdobyć, powinniśmy opisać grę w formie akapitowej.

Gra deszczowa

Przedmiotem tej gry jest łapanie kropel deszczu, zanim uderzą o ziemię. Co jakiś czas (w zależności od poziomu trudności), nowa kropla spada z górnej części ekranu w losowym położeniu poziomym z losową prędkością pionową. Gracz musi łapać krople deszczu za pomocą myszy, aby nie dopuścić do przedostania się kropel deszczu na dół ekranu.

Ćwiczenie 10.1: Napisz pomysł na projekt, który chcesz stworzyć. Niech to będzie proste, ale nie za proste. Kilka elementów, zrobi kilka zachowań.

Zobaczmy teraz, czy możemy wziąć pomysł "Rain Game" i podzielić go na mniejsze części. Jak to robimy? Po pierwsze, możemy zacząć od myślenia o elementach gry: kroplach deszczu i łapaczu. Po drugie, powinniśmy myśleć o zachowaniach tych elementów. Na przykład będziemy potrzebować mechanizmu taktowania, aby krople spadały "tak często". Będziemy również musieli określić, kiedy pojawia się kropla deszczu "Złapana "Organizujmy te części bardziej formalnie.

Część 1. Opracuj program z okręgiem kontrolowanym przez mysz. To koło będzie sterowanym przez użytkownika "łapaczem deszczu". "

Część 2. Napisz program, aby sprawdzić, czy dwa okręgi się przecinają. Zostanie użyty do ustalenia, czy łapacz deszczu złapał kroplę deszczu.

Część 3. Napisz program czasowy, który wykonuje funkcję co N sekund.

Część 4. Napisz program z okręgami spadającymi z góry ekranu na dół. To będą krople deszczu.

Części od 1 do 3 są proste i można je wykonać za jednym zamachem. Jednak w przypadku Części 4, mimo że stanowi ona jeden element większego projektu, jest na tyle złożona, że będziemy musieli wykonać to dokładne ćwiczenie, dzieląc je na mniejsze etapy i budując je z powrotem.

Ćwiczenie 10-2: Weź swój pomysł z ćwiczenia 10-1 i zapisz poszczególne części. Staraj się, aby części były tak proste, jak to tylko możliwe (prawie do tego stopnia, że wydaje się absurdalne). Jeśli części są zbyt skomplikowane, rozbijaj je jeszcze bardziej.

Sekcje od 10.4 do 10.7 będą następowały po krokach 2a, 2b i 2c (patrz pole breakout na s. 166) dla każdej poszczególnej części. Dla każdej części najpierw opracujemy algorytm pseudokodowania, następnie w aktualnym kodzie i zakończymy wersją obiektową. Jeśli wykonamy swoją pracę poprawnie, cała wymagana funkcjonalność zostanie wbudowana w klasę, która następnie może być łatwo skopiowana do samego projektu końcowego, gdy dojdziemy do Kroku 3 (integracja wszystkich części)

10.4 Część 1: Łapacz

Jest to najprostsza część do skonstruowania i wymaga niewiele więcej niż to, czego nauczyliśmy się w części 3. Posiadanie pseudokodu, który ma tylko dwie linie długie, jest dobrym znakiem, wskazując, że ten krok jest wystarczająco mały, aby obsłużyć i nie trzeba go przekształcać w parzyste. mniejsze części.

Pseudo kod:

• Wymaż tło.

• Narysuj elipsę w miejscu myszy.

Przetłumaczenie go na kod jest łatwe:

```
void setup() {

size(400,400);

smooth();

}

void draw() {

background(255);

stroke(0);

fill(175);

ellipse(mouseX,mouseY,64,64);

}
```

To dobry krok, ale nie skończyliśmy. Jak stwierdzono, naszym celem jest opracowanie programu do zbierania deszczu w sposób zorientowany obiektowo. Kiedy przyjmiemy ten kod i uwzględnimy go w ostatecznym programie, będziemy chcieli, aby został on podzielony na klasy, abyśmy mogli wykonać obiekt "łapacza". Nasz pseudokod zostałby zmieniony tak, by wyglądał następująco.

Setup

• Zainicjuj obiekt przyciągania.

Rysowanie:

• Wymaż tło.

• Ustaw lokalizację łapacza na lokalizację myszy.

• Wyświetl Łapacza.

Przykład 10-1 pokazuje przepisany kod przy założeniu obiektu Łapacza

Przykład 10-1:Łapacz

```
Catcher catcher;
void setup() {
size(400,400);
catcher = new Catcher(32);
smooth();
}
void draw() {
background(255);
catcher.setLocation(mouseX,mouseY);
catcher.display();
}
```

Klasa Catcher jest raczej prosta, ze zmiennymi dotyczącymi lokalizacji i rozmiaru oraz dwiema funkcjami, jedną do ustawienia lokalizacji i jedną do wyświetlenia.

```
class Catcher {
float r; // radius
float x,y; // location
Catcher(float tempR) {
r = tempR;
x = 0;
y = 0;
```

```
}

void setLocation(float tempX, float tempY) {

x = tempX;

y = tempY;

}

void display() {

stroke(0);

fill(175);

ellipse(x,y,r*2,r*2);

}

}
```

10.5 Część 2: Przecięcie

Część 2 naszej listy wymaga od nas określenia, kiedy łapacz i kropla deszczu przecinają się. Funkcja przecięcia jest tym, na czym chcemy się skupić na rozwoju na tym etapie. Zaczniemy od prostej klasy piłki odbijającej (którą widzieliśmy w Przykładzie 5-6) i ustalimy, kiedy krzyżują się dwa podskakujące koła. Podczas procesu "integracji" ta funkcja przecinania () zostanie włączona do modułu klasy Łapacz, aby złapać krople deszczu. Oto nasz algorytm dla tej części przecięcia.

Setup:

• Utwórz dwa obiekty kulkowe.

Rysowanie:

• Przesuń piłki.

• Jeśli piłka # 1 przecina kulę # 2, zmień kolor obu kul na biały. W przeciwnym razie pozostaw kolor szary.

• Wyświetlaj kule.

Z pewnością ciężką pracą jest test przecięcia, który za chwilę osiągniemy. Po pierwsze, tutaj jest to, czego potrzebujemy do prostej podskakującej klasy "Ball" bez testu przecięcia.

Dane:

• Lokalizacja X i Y.

• Promień.

• Prędkość w kierunkach X i Y.

Funkcje:

• Konstruktor.

- Ustaw promień na podstawie argumentu

- Wybierz losową lokalizację.

- Wybierz prędkość losową.

• Ruszaj się.

- Przyrost X przez prędkość w kierunku X.

- Zwiększ Y przy prędkości w kierunku Y.

- Jeśli Ball uderzy w jakąkolwiek krawędź, odwróć kierunek.

• Wyświetl.

- Narysuj okrąg w miejscu X i Y.

Jesteśmy teraz gotowi do przetłumaczenia tego na kod.

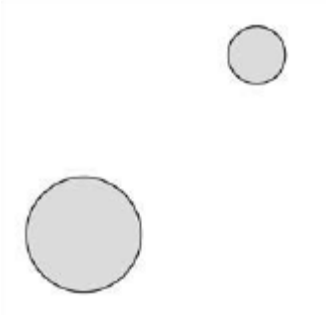
```
class Ball {
float r; // radius
float x,y; // location
float xspeed,yspeed; // speed
// Constructor
Ball(float tempR) {
r = tempR;
x = random(width);
y = random(height);
xspeed = random( -5,5);
yspeed = random( -5,5);
}
void move() {
x + = xspeed; // Increment x
y + = yspeed; // Increment y
// Check horizontal edges
if (x > width | | x < 0) {
xspeed * = - 1;
}
//Check vertical edges
if (y > height | | y < 0) {
yspeed * = – 1;
```

```
}

}

// Draw the ball

void display() {

stroke(255);

fill(100,50);

ellipse(x,y,r*2,r*2);

}

}
```

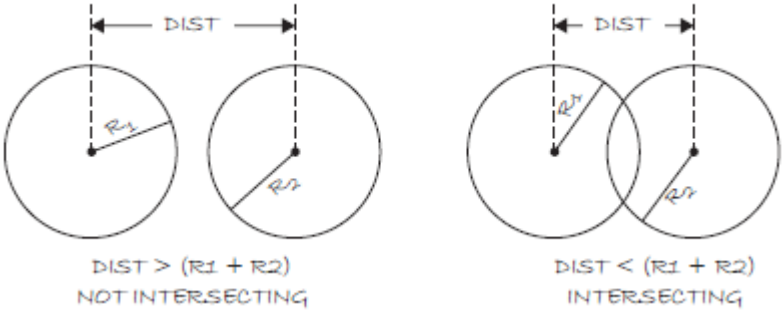
Stąd całkiem łatwo jest stworzyć szkic z dwoma obiektami kulkowymi. Ostatecznie, w ostatecznym szkicu, potrzebujemy tablicy dla wielu kropel deszczu, ale na razie dwie zmienne kulkowe będą prostsze.

Przykład 10-2 Łapacz (cd.): Dwie kulki

```
// Two ball variables

Ball ball1;

Ball ball2;

void setup() {

size(400,400);

smooth();

// Initialize balls

ball1 = new Ball(64);

ball2 = new Ball(32);

}

void draw() {

background(0);

// Move and display balls

ball1.move();

ball2.move();

ball1.display();

ball2.display();

}
```



Po skonfigurowaniu naszego systemu do poruszania się po ekranie dwoma okręgami, musimy opracować algorytm do określania, czy okręgi się przecinają. W Processing wiemy, że możemy obliczyć odległość między dwoma punktami za pomocą funkcji dist (). Mamy również dostęp do promienia każdego koła (zmienna r wewnątrz każdego obiektu). Diagram na rysunku poniżej pokazuje, w jaki sposób możemy porównać odległość między okręgami a sumą promieni, aby określić, czy okręgi zachodzą na siebie.



OK, więc zakładając, że:

• x_1, y_1: współrzędne okręgu pierwszego

• x_2, y_2: współrzędne okręgu drugiego

• r_1: promień okręgu pierwszego

• r_2: promień koła drugiego

Mamy oświadczenie:

Jeśli odległość między (x_1, y_1) i (x_2, y_2) jest mniejsza niż suma r_1 i r_2, koło jeden przecina okrąg dwa. Naszym zadaniem jest teraz napisać funkcję, która zwraca wartość true lub false w oparciu o powyższą instrukcję.

```
// A function that returns true or false based on whether two circles intersect

// If distance is less than the sum of radii the circles touch

boolean intersect(float x1, float y1, float x2, float y2, float r1, float r2) {

float distance = dist(x1,y2,x2,y2); // Calculate distance

if (distance < r1 + r2) { // Compare distance to r1 + r2

return true;
```

```
} else {
return false;
}
}
```

Teraz, gdy funkcja jest zakończona, możemy przetestować ją z danymi z ball1 i ball2.

```
boolean intersecting =  intersect(ball1.x,ball1.y,ball2.x,ball2.y,ball1.r,ball2.r);
if (intersecting) {
println( "The circles are intersecting! ");
}
```

Powyższy kod jest nieco niewygodny i warto będzie skorzystać z funkcji o jeden krok dalej, włączając ją do samej klasy piłki. Spójrzmy najpierw na cały główny program w jego obecnym stanie.

```
// Two ball variables
Ball ball1;
Ball ball2;
void setup() {
size(400,400);
framerate(30);
smooth();
// Initialize balls
ball1 = new Ball(64);
ball2 = new Ball(32);
}
void draw() {
background(0);
// Move and display balls
ball1.move();
ball2.move();
ball1.display();
ball2.display();
boolean intersecting = intersect(ball1.x,ball1.y,ball2.x,ball2.y,ball1.r,ball2.r);
if (intersecting) {
```

println(“The circles are intersecting! ”);

}

}

// A function that returns true or false based on whether two circles intersect

// If distance is less than the sum of radii the circles touch

boolean intersect(float x1, float y1, float x2, float y2, float r1, float r2) {

float distance = dist(x1,y2,x2,y2); // Calculate distance

if (distance < r1 + r2) { // Compare distance to r1 + r2

return true;

} else {

return false;

}

}

Ponieważ zaprogramowaliśmy kulki w sposób zorientowany na obiekt, nie jest szalenie logiczne, że nagle mamy funkcję przecięcia (), która żyje poza klasą piłki. Obiekt kulowy powinien wiedzieć, jak przetestować, czy przecina inny obiekt kulki. Nasz kod może zostać ulepszony poprzez włączenie logiki przecięcia do samej klasy, mówiąc "ball1.intersect (ball2); "Czy Ball 1 przecina kulę 2?

void draw() {

background(0);

// Move and display balls

ball1.move();

ball2.move();

ball1.display();

ball2.display();

boolean intersecting = ball1.intersect(ball2); // Zakłada, że funkcja intersect () zawiera klasę Ball, która
//zwraca wartość true lub false.

if (intersecting) {

println(" The circles are intersecting! ");

}

}

Po tym modelu i algorytmie do testowania przecięcia, oto funkcja wewnątrz samej klasy piłki. Zwróć uwagę, że funkcja korzysta zarówno z własnej lokalizacji (x i y), jak i z położenia innej kuli (b.x i b.y).

// A function that returns true or false based on whether two Ball objects intersect

```
// If distance is less than the sum of radii the circles touch
boolean intersect(Ball b) {
float distance = dist(x,y,b.x,b.y); // Calculate distance
if (distance < r + b.r) { // Compare distance to sum of radii
return true;
} else {
return false;
}
}
```

Łącząc to wszystko razem, mamy kod z przykładu 10-3.

Przykład 10-3: Odbijająca się piłka ze skrzyżowaniem

```
// Two ball variables
Ball ball1;
Ball ball2;
void setup() {
size(400,400);
smooth();
// Initialize balls
ball1 = new Ball(64);
ball2 = new Ball(32);
}
void draw() {
background(255);      // Nowy! Obiekt może mieć funkcję, która przyjmuje inny obiekt jako
                      //argument. Jest to jeden ze sposobów komunikacji obiektów. W tym
                      //przypadku sprawdzają, czy się przecinają.
// Move and display balls
ball1.move();
ball2.move();
if (ball1.intersect(ball2)) {
ball1.highlight();
ball2.highlight();
}
```

```
ball1.display();
ball2.display();
}
class Ball {
float r; // radius
float x,y;
float xspeed,yspeed;
color c = color(100,50);
// Constructor
Ball(float tempR) {
r = tempR;
x = random(width);
y = random(height);
xspeed = random( –5,5);
yspeed = random( –5,5);
}
void move() {
x + = xspeed; // Increment x
y + = yspeed; // Increment y
// Check horizontal edges
if (x > width | | x < 0) {
xspeed * = – 1;
}
// Check vertical edges
if (y > height | | y < 0) {
yspeed * = – 1;
}
}
void highlight() { // Ilekroć kule się stykają, wywoływana jest ta funkcja highlight (), a kolor jest
                    //przyciemniony.
c = color(0,150);
```

```
}

// Draw the ball

void display() {

stroke(0);

fill(c);

ellipse(x,y,r*2,r*2); // Po wyświetleniu kulki kolor powraca do ciemniejszej szarości.

c = color(100,50);

}

// A function that returns true or false based on whether two circles intersect

// If distance is less than the sum of radii the circles touch

boolean intersect(Ball b) {   // Obiekty można również przekazywać do funkcji jako argumenty!

float distance = dist(x,y,b.x,b.y); // Calculate distance

if (distance < r + b.r) { // Compare distance to

sum of radii

return true;

} else {

return false;

}

}

}
```

10.6 Część 3: Timer

Naszym następnym zadaniem jest opracowanie timera, który wykonuje funkcję co N sekund. Znowu zrobimy to w dwóch krokach, najpierw za pomocą głównej treści programu, a po drugie, biorąc logikę i umieszczając ją w klasie Timer. Przetwarzanie ma funkcje hour(), second(), minute(), month(), day () i year (), aby poradzić sobie z czasem. Możemy użyć funkcji second (), aby określić, ile czasu minęło. Jednak nie jest to bardzo wygodne, ponieważ second() przewraca się z 60 na 0 na końcu każdej minuty. W celu utworzenia timera najlepiej jest użyć funkcji millis (). Przede wszystkim millis (), który zwraca liczbę milisekundy od rozpoczęcia szkicu pozwalają na znacznie większą precyzję. Jedna milisekunda to jedna część wieczna (1000 ms - 1 s). Po drugie, millis() nigdy nie cofa się do zera, więc pyta o milisekundy w jednej chwili i odjęcie od milisekund w późniejszym czasie zawsze będą skutkować upływem czasu. Powiedzmy, że chcemy, aby aplet zmieniał kolor tła na czerwony po 5 sekundach od rozpoczęcia. Pięć sekund to 5000 ms, więc łatwo jest sprawdzić, czy wynik funkcji millis() jest większy niż 5000.

```
if (millis ()> 5000) {

tło (255,0,0);
```

}

Sprawiając, że problem jest nieco bardziej skomplikowany, możemy rozszerzyć program, aby zmienić tło na nowy losowy kolor co 5 sekund.

Setup:

• Oszczędź czas przy starcie (pamiętaj, że zawsze powinno być zero, ale mimo to warto zachować go w zmiennej). Nazwij to "savedTime".

Rysowanie:

• Oblicz czas upływający jako bieżący czas (tj. Millis ()) minus zapisany czas. Zapisz to jako "czas przeszły".

• Jeśli wartość parametru min. Jest większa niż 5000, wypełnij nowe losowe tło i zresetuj zapisany czas do bieżącego czasu. Ten krok spowoduje ponowne uruchomienie timera.

Przykład 10-4 tłumaczy to na kod.

Przykład 10-4: Wdrażanie licznika czasu

```
int savedTime;

int totalTime = 5000;

void setup() {

size(200,200);

background(0);

savedTime = millis();

}

void draw() {

// Calculate how much time has passed

int passedTime = millis() - savedTime;

// Has five seconds passed?

if (passedTime > totalTime) {

println( " 5 seconds have passed! " );

background(random(255)); // Color a new background

savedTime = millis(); // Save the current time to restart the timer!

}

}
```

Po opracowaniu powyższej logiki możemy teraz przenieść stoper na klasę. Zastanówmy się, jakie dane są zaangażowane w timer. Zegar musi znać czas, w którym się rozpoczął (zapisany czas) i czas jego działania (totalTime).

Setup:

• zapisany czas

• czas całkowity

Zegar musi również być w stanie uruchomić, a także sprawdzić i sprawdzić, czy jest gotowy.

Funkcje:

• początek( )

• isFinished () - zwraca true lub false

Biorąc kod z przykładu nieobiektowego i budując go z powyższą strukturą, mamy kod pokazany w Przykładzie 10-5.

Przykład 10-5: Licznik czasu zorientowany na obiekt

```
Timer timer;

void setup() {

size(200,200);

background(0);

timer = new Timer(5000);

timer.start();

}

void draw() {

if (timer.isFinished()) {

background(random(255));

timer.start();

}

}

class Timer {

int savedTime; // When Timer started

int totalTime; // How long Timer should last

Timer(int tempTotalTime) {

totalTime = tempTotalTime;

}

// Starting the timer

void start() {  // Po uruchomieniu timera przechowuje bieżący czas w milisekundach.
```

```
savedTime = millis();

}

boolean isFinished() {    // Funkcja isFinished () zwraca wartość true, jeśli minęło 5000 ms. Praca timera
                          //jest hodowana do tej metody.

// Check how much time has passed

int passedTime = millis()- savedTime;

if (passedTime > totalTime) {

return true;

} else {

return false;

}

}

}
```

10.7 Część 4: Krople deszczu

Już prawie jesteśmy. Stworzyliśmy Łapacza, wiemy, jak przetestować przecięcie i ukończyliśmy obiekt timera. Ostatnim elementem układanki są same krople deszczu. W końcu chcemy tablicy obiektów Raindrop spadających z górnej części okna na dół. Ponieważ ten krok polega na tworzeniu szeregu poruszających się obiektów, użyteczne jest podejście do tej czwartej części jako serii jeszcze mniejszych kroków, podpunktów części 4, ponowne myślenie o poszczególnych elementach i zachowaniach, których będziemy potrzebować.

Część 4 Podczęści:

Część 4.1. Pojedyncza poruszająca się kropla deszczu.

Część 4.2. Tablica obiektów z kroplami deszczu.

Część 4.3. Elastyczna liczba kropel deszczu (pojawiająca się pojedynczo).

Część 4.4. Ciekawy wygląd kropli deszczu.

Część 4.1, tworzenie ruchu kropli deszczu (prosty krąg na razie) jest łatwa - Rozdział 3 jest łatwy.

• Zwiększ wartość kropli deszczu y.

• Wyświetl kroplę deszczu.

W tłumaczeniu na kod mamy Część 4.1 - Pojedyncza poruszająca się kropla deszczu, pokazana w Przykładzie 10-6

Przykład 10-6: Proste zachowanie kropel deszczu

```
float x,y; // Variables for drop location

void setup() {

size(400,400);
```

```
background(0);

x = width/2;

y = 0;

}

void draw() {

background(255);

// Display the drop

fill(50,100,150);

noStroke();

ellipse(x,y,16,16);

// Move the drop

y + + ;

}
```

Znowu jednak musimy pójść o krok dalej i stworzyć klasę Drop - w końcu ostatecznie będziemy potrzebować szeregu kropli. Tworząc klasę, możemy dodać kilka dodatkowych zmiennych, takich jak prędkość i rozmiar, a także funkcję do sprawdzenia, czy kropla deszczu dociera do dolnej części ekranu, co przydaje się później do oceny gry.

```
class Drop {

float x,y; // Variables for location of raindrop

float speed; // Speed of raindrop

color c;  // Obiekt z kroplami deszczu ma lokalizację, szybkość, kolor i rozmiar

float r; // Radius of raindrop

Drop() {

r = 8; // All raindrops are the same size

x = random(width); // Start with a random x location

y = -r*4; // Start a little above the window

speed = random(1,5); // Pick a random speed

c = color(50,100,150); // Color

}

//Move the raindrop down

void move() {  // Inkrementacja y jest teraz w funkcji move ().

y + = speed; // Increment by speed
```

```
}

// Check if it hits the bottom

boolean reachedBottom() { // Ponadto mamy funkcję, która określa, czy kropla opuszcza okno.

// If we go a little past the bottom

if (y > height + r*4) {

return true;

} else {

return false;

}

}

// Display the raindrop

void display() {

// Display the drop

fill(50,100,150);

noStroke();

ellipse(x,y,r*2,r*2);

}

}
```

Zanim przejdziemy do Części 4.3, tablicy kropli, powinniśmy się upewnić, że pojedynczy obiekt typu Drop działa poprawnie. Jako ćwiczenie wypełnij kod z ćwiczenia 10-3, który przetestowałby pojedynczy obiekt upuszczenia

Ćwiczenie 10-3: Wypełnij puste pola poniżej, wykonując szkic "kropla testowa".

```
Drop drop;

void setup() {

size(200,200);

_____;

}

void draw() {

background(255);

drop. _____;

_____;

}
```

Teraz, gdy jest to ukończone, następnym krokiem jest przejście z jednej kropli do tablicy kropli - część 4.2. To właśnie technika, którą udoskonaliliśmy w części 9.

```
// An array of drops

Drop[] drops = new Drop[50];  // Zamiast jednego obiektu Drop, tablica 50.

void setup() {

size(400,400);

smooth();

// Initialize all drops

for (int i = 0; i < drops.length; i + + ) { // Używanie pętli do inicjowania wszystkich kropli

drops[i] = new Drop();

}

}

void draw() {

background(255);

// Move and display all drops

for (int i = 0; i < drops.length; i + + ) { // Przenieś i wyświetl wszystkie krople.

drops[i].move();

drops[i].display();

}

}
```

Problem z powyższym kodem polega na tym, że krople deszczu pojawiają się wszystkie naraz. Zgodnie ze specyfikacjami, które przygotowaliśmy dla naszej gry, chcemy, aby krople deszczu pojawiały się pojedynczo, co N sekund – jesteśmy teraz w części 4.3 - Elastyczna liczba kropel deszczu (pojawiających się pojedynczo). Możemy teraz pominąć martwienie się o zegar i po prostu pojawi się jedna nowa kropla deszczu pojawiająca się w każdej klatce. Powinniśmy również sprawić, by nasz układ był znacznie większy, pozwalając na dużo więcej kropel deszczu. Aby to działało, potrzebujemy nowej zmiennej, aby śledzić całkowitą liczbę kropli - "totalDrops". Większość przykładowych przykładów obejmuje chodzenie po całej tablicy, aby poradzić sobie z całą listą. Teraz chcemy uzyskać dostęp do części listy, liczby przechowywanej w totalDrops. Napiszmy trochę pseudokodu, aby opisać ten proces:

Setup:

• Stwórz tablicę kropli z 1000 spacji.

• Ustaw totalDrops = 0.

Rysowanie:

• Utwórz nowy spadek w tablicy (w miejscu totalDrops). Ponieważ totalDrops zaczyna się od 0, najpierw stworzymy nową kroplę deszczu na pierwszym miejscu tablicy.

• Zwiększ sumaryczne Wpływy (aby następnym razem przybyć tutaj, utworzymy kroplę w następnym miejscu w tablicy).

• Jeśli totalDrops przekracza rozmiar tablicy, zresetuj ją do zera i rozpocznij od nowa.

• Przenieś i wyświetl wszystkie dostępne kropelki (tj. TotalDrops).

Przykład 10-7 tłumaczy powyższy pseudokod na kod.

Przykład 10-7: Krople pojedynczo

```
// An array of drops

Drop[] drops = new Drop[1000];

int totalDrops = 0; // Nowa zmienna do śledzenia całkowitej liczby kropli, których chcemy użyć!

void setup() {

size(400,400);

smooth();

background(0);

}

void draw() {

background(255);

// Initialize one drop

drops[totalDrops] = new Drop();

// Increment totalDrops

totalDrops + + ;

// If we hit the end of the array

if (totalDrops > = drops.length) {

totalDrops = 0; //Start over

}

// Move and display drops

for (int i = 0; i < totalDrops; i + + ) {        // Nowy! Nie poruszamy się już i wyświetlamy wszystkie dropy,
                                                 //a raczej tylko "totalDrops", które są obecnie obecne w grze.

drops[i].move();

drops[i].display();

}

}
```

Poświęciliśmy czas, aby dowiedzieć się, jak chcemy, aby kropla deszczu się poruszała, stworzyła klasę, która wykazuje takie zachowanie i stworzyła szereg obiektów z tej klasy. Cały czas jednak używaliśmy kółka do wyświetlania kropli. Zaletą tego jest to, że byliśmy w stanie opóźnić obawy dotyczące kodu rysunku i skupić się na zachowaniach ruchu i organizacji danych i funkcji. Teraz możemy skupić się na tym, jak wyglądają krople - Część 4.4-Finalizowanie wyglądu kropli deszczu. Jednym ze sposobów stworzenia bardziej "przypominającego kroplę" wyglądu jest narysowanie sekwencji okręgów w kierunku pionowym, zaczynających się od małych i coraz większych podczas ich przesuwania w dół.

Przykład 10-8 : hodowcy wyglądająca kropla deszczu

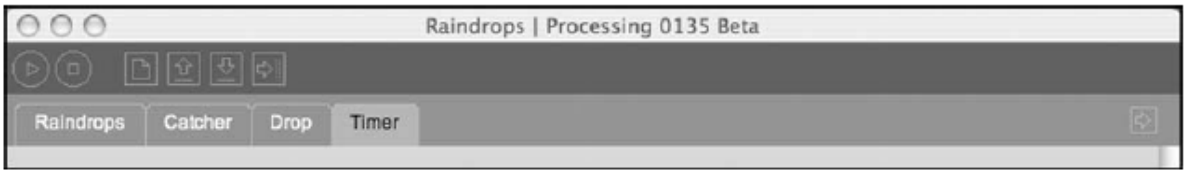
```
background(255);

for (int i = 2; i < 8; i + + ) {

noStroke();

fill(0);

ellipse(width/2,height/2 + i*4,i*2,i*2);

}
```

Możemy włączyć ten algorytm do klasy kropli deszczu z przykładu 10-7, używając x i y dla początku lokalizacji elipsy, a promień kropli jako maksymalną wartość dla i w pętli for. Dane wyjściowe pokazano na rysunku

```
// Display the raindrop

void display() {

// Display the drop

noStroke();

fill(c);

for (int i = 2; i < r; i + + ) {

ellipse(x,y + i*4,i*2,i*2);

}

}
```

10.8 Integracja: Puttin 'on the Ritz

Już czas. Po opracowaniu poszczególnych elementów i upewnieniu się, że każdy z nich działa poprawnie, możemy połączyć je w jeden program. Pierwszym krokiem jest stworzenie nowego szkicu przetwarzania, który ma cztery zakładki, po jednym dla każdej z naszych klas i jednego głównego programu, jak pokazano na rysunku



Pierwszym krokiem jest skopiowanie i wklejenie kodu każdej klasy do każdej karty. Indywidualnie, nie będą musieli się zmieniać, więc nie musimy ponownie wracać do kodu. To, co musimy zrobić, to powrót do głównego programu - co dzieje się w setup () i draw (). Odnosząc się do pierwotnego opisu gry i wiedząc, w jaki sposób zostały złożone, możemy napisać algorytm pseudokodowania dla całej gry.

Setup:

• Utwórz obiekt Catchera.

• Utwórz szereg kropli.

• Ustaw totalDrops równy 0.

• Utwórz obiekt Timer.

• Uruchom timer.

Rysowanie:

• Ustaw lokalizację Catchera na położenie myszy.

• Wyświetl Catchera.

• Przenieś wszystkie dostępne Krople.

• Wyświetl wszystkie dostępne Drops.

• Jeśli Łapacz przecina jakikolwiek Kroplę.

- Usuń Drop z ekranu.

• Jeśli timer się zakończył:

- Zwiększ liczbę kropli.

- Uruchom ponownie timer.

Zauważ, że każdy pojedynczy krok w powyższym programie został już opracowany wcześniej w części z jednym wyjątkiem: "Usuń zrzut z ekranu. "To jest raczej powszechne. Nawet z przełamaniem pomysłu na części i rozpracowywanie ich pojedynczo, można pominąć małe kawałki. Na szczęście ten element funkcjonalności jest dość prosty i przy odrobinie pomysłowości zobaczymy, jak możemy go wsunąć podczas montażu. Jednym ze sposobów podejścia do montażu powyższego algorytmu jest rozpoczęcie od połączenia wszystkich elementów w jeden szkic i nie martwienie się o to, jak wchodzą w interakcje. Innymi słowy, wszystko oprócz czasu, w którym włącza się timer, powoduje wyrzucanie kropli i testowanie skrzyżowania. Aby to osiągnąć, wystarczy skopiować / wkleić ze zmiennych globalnych

każdej części, setup() i draw()! Oto zmienne globalne: obiekt Catcher, tablica obiektów typu Drop, obiekt Timer i liczba całkowita do przechowywania całkowitej liczby kropli.

```
Catcher catcher; // One catcher object

Timer timer; // One timer object

Drop[] drops; // An array of drop objects

int totalDrops = 0; // totalDrops
```

W setup () zmienne są inicjowane. Pamiętaj jednak, że możemy pominąć inicjowanie poszczególnych kropli w tablicy, ponieważ będą one tworzone pojedynczo. Będziemy musieli również wywołać funkcję start () timera.

```
void setup() {

size(400,400);

catcher = new Catcher(32); // Create the catcher with a radius of 32

drops = new Drop[1000]; // Create 1000 spots in the array

timer = new Timer(2000); // Create a timer that goes off every 2 seconds

timer.start(); // Starting the timer

}
```

W draw () obiekty wywołują swoje metody. Ponownie, właśnie robimy kod z każdej części, którą zrobiliśmy oddzielnie wcześniej w tym rozdziale i wklejamy w sekwencji.

Przykład 10-9: Używanie wszystkich obiektów w jednym szkicu

```
Catcher catcher; // One catcher object

Timer timer; // One timer object

Drop[] drops; // An array of drop objects

int totalDrops = 0; // totalDrops

void setup() {

size(400,400);

smooth();

catcher = new Catcher(32); // Create the catcher with a radius of 32

drops = new Drop[1000]; // Create 1000 spots in the array

timer = new Timer(2000); // Create a timer that goes off every 2 seconds

timer.start(); // Starting the timer

}

void draw() {
```

```
background(255);
catcher.setLocation(mouseX,mouseY); // Set catcher location
catcher.display(); // Display the catcher // Z części 1. The Catcher!

// Check the timer
if (timer.isFinished()) {  // Z części 3. Timer!
println( " 2 seconds have passed! " );
timer.start();
}

// Deal with raindrops
// Initialize one drop
drops[totalDrops] = new Drop();
//Increment totalDrops
totalDrops + + ;  // Z części 4. Krople deszczu!
// If we hit the end of the array
if (totalDrops > = drops.length) {
totalDrops = 0; // Start over
}
// Move and display all drops
for (int i = 0; i < totalDrops; i + + ) {
drops[i].move();
drops[i].display();
}
}
```

Następnym krokiem jest rozwinięcie tych koncepcji i sprawienie, by działały razem. Na przykład, powinniśmy utworzyć nową kroplę deszczu dopiero po upływie dwóch sekund (jak wskazuje funkcja isFinished() timera).

```
// Check the timer
if (timer.isFinished()) {
```

```
// Deal with raindrops          // Obiekty pracujące razem! Tutaj, gdy licznik czasu "zostanie
                                //zakończony", dodawany jest obiekt Upuść (poprzez
                                //inkrementowanie "totalDrops").

// Initialize one drop

drops[totalDrops] = new Drop();

// Increment totalDrops

totalDrops + + ;

// If we hit the end of the array

if (totalDrops > = drops.length) {

totalDrops = 0; // Start over

}

timer.start();

}
```

Musimy również ustalić, kiedy obiekt Catchera przecina kroplę. W rozdziale 10.5 przetestowaliśmy pod kątem przecięcia wywołując funkcję intersect (), którą napisaliśmy wewnątrz klasy Ball.

```
boolean intersecting = ball1.intersect(ball2);

if (intersecting) {

println( "The circles are intersecting! ");

}
```

Możemy zrobić to samo tutaj, wywołując funkcję intersect () w klasie Catchera i przechodząc przez każdą kroplę deszczu w systemie. Zamiast drukować wiadomość, będziemy chcieli wpłynąć na samą kroplę deszczu, mówiąc, że może zniknąć. Kod jest założeniem, że funkcja catch () wykona zadanie.

```
// Move and display all drops

for (int i = 0; i < totalDrops; i + + ) {

drops[i].move();

drops[i].display();

if (catcher.intersect(drops[i])) {

drops[i].caught();      // Obiekty pracujące razem! Tutaj obiekt Catcher sprawdza, czy przecina
                        //dowolny obiekt Drop w tablicy kropel.

}

}
```

Nasz obiekt Catcher początkowo nie zawierał funkcji intersect (), ani Drop nie zawierał catch (). Oto kilka nowych funkcji, które będziemy musieli napisać w ramach procesu integracji. intersect () jest

łatwe do wprowadzenia, ponieważ rozwiązaliśmy problem już w rozdziale 10.5 i możemy dosłownie skopiować go do klasy Catcher (zmieniając argument z obiektu Ball na obiekt Drop).

```
// A function that returns true or false based if the catcher intersects a raindrop

boolean intersect(Drop d) {

// Calculate distance

float distance = dist(x,y,d.x,d.y);

// Compare distance to sum of radii     // Oprócz funkcji wywołujących możemy uzyskać dostęp do
                                        //zmiennych wewnątrz obiektu za pomocą składni kropkowej.

if (distance < r + d.r) {

return true;

} else {

return false;

}

}
```

Po upuszczeniu kropli, ustawimy jego lokalizację gdzieś poza ekranem (tak, aby nie było widać, że jest to odpowiednik "znikania") i zatrzymujemy ruch, ustawiając prędkość równą 0. Chociaż nie wyszło Ta funkcjonalność przed procesem integracji jest na tyle prosta, że można ją teraz wrzucić.

```
// If the drop is caught

void caught() {

speed = 0; // Stop it from moving by setting speed equal to zero

y = – 1000; // Set the location to somewhere way off-screen

}
```

Skończyliśmy! Dla odniesienia, Przykład 10-10 jest całym szkicem. Zegar jest zmieniany tak, aby był uruchamiany co 300 ms, przez co gra była nieco trudniejsza.

Przykład 10-10: Gra w łapanie kropli deszczu

```
Catcher catcher; // One catcher object

Timer timer; // One timer object

Drop[] drops; // An array of drop objects

int totalDrops = 0; // totalDrops

void setup() {

size(400,400);

smooth();

catcher = new Catcher(32); // Create the catcher with a radius of 32
```

```
drops = new Drop[1000]; // Create 1000 spots in the array
timer = new Timer(300); // Create a timer that goes off every 2 seconds
timer.start(); // Starting the timer
}
void draw() {
background(255);
catcher.setLocation(mouseX,mouseY); // Set catcher location
catcher.display(); // Display the catcher
// Check the timer
if (timer.isFinished()) {
// Deal with raindrops
// Initialize one drop
drops[totalDrops] = new Drop();
// Increment totalDrops
totalDrops + + ;
// If we hit the end of the array
if (totalDrops > = drops.length) {
totalDrops = 0; // Start over
}
timer.start();
}
// Move and display all drops
for (int i = 0; i < totalDrops; i + + ) {
drops[i].move();
drops[i].display();
if (catcher.intersect(drops[i])) {
drops[i].caught();
}
}
}
class Catcher {
```

```
float r; // radius
color col; // color
float x,y; // location
Catcher(float tempR) {
r = tempR;
col = color(50,10,10,150);
x = 0;
y = 0;
}
void setLocation(float tempX, float tempY) {
x = tempX;
y = tempY;
}
void display() {
stroke(0);
fill(col);
ellipse(x,y,r*2,r*2);
}
// A function that returns true or false based if the catcher intersects a raindrop
boolean intersect(Drop d) {
float distance = dist(x,y,d.x,d.y); // Calculate distance
if (distance < r + d.r) { // Compare distance to sum of radii
return true;
} else {
return false;
}
}
}
class Drop {
float x,y; // Variables for location of raindrop
float speed; // Speed of raindrop
```

```
color c;
float r; // Radius of raindrop
Drop() {
r = 8; // All raindrops are the same size
x = random(width); // Start with a random x location
y = -r*4; // Start a little above the window
speed = random(1,5); // Pick a random speed
c = color(50,100,150); // Color
}
// Move the raindrop down
void move() {
y + = speed; // Increment by speed
}
// Check if it hits the bottom
boolean reachedBottom() {
if (y > height + r*4) { // If we go a little beyond the bottom
return true;
} else {
return false;
}
}
// Display the raindrop
void display() {
// Display the drop
fill(c);
noStroke();
for (int i = 2; i < r; i + + ) {
ellipse(x,y + i*4,i*2,i*2);
}
}
// If the drop is caught
```

```
void caught() {
speed = 0; // Stop it from moving by setting speed equal to zero
// Set the location to somewhere way off-screen
y = – 1000;
}
}
class Timer {
int savedTime; // When Timer started
int totalTime; // How long Timer should last
Timer(int tempTotalTime) {
totalTime = tempTotalTime;
}
// Starting the timer
void start() {
savedTime = millis();
}
boolean isFinished() {
// Check out much time has passed
int passedTime = millis()- savedTime;
if (passedTime > totalTime) {
return true;
} else {
return false;
}
}
}
```

Ćwiczenie 10-4: Zaimplementuj system oceny gry. Uruchom gracza z 10 punktami. Za każdą kroplę deszczu, która osiągnie dno, zmniejsz wynik o 1. Jeśli wszystkie 1000 kropel deszczu spadną bez uzyskania wyniku do zera, rozpoczyna się nowy poziom, a krople deszczu pojawiają się szybciej. Jeśli 10 kropel deszczu osiąga dno podczas dowolnego poziomu, gracz przegrywa. Pokaż wynik na ekranie jako prostokąt, który zmniejsza swój rozmiar. Nie próbuj implementować wszystkich tych funkcji naraz. Wykonuj je krok po kroku!

10.9 Przygotowanie do aktu II

Celem tej części nie jest nauczenie się programowania gry polegającej na łapaniu spadających kropel deszczu, a raczej rozwijanie podejścia do rozwiązywania problemów - branie pomysłu, rozbijanie go na części, rozwijanie pseudokodu dla tych części i wdrażanie ich. jeden bardzo mały krok na raz. Należy pamiętać, że przyzwyczajenie się do tego procesu wymaga czasu i wymaga praktyki. Każdy stara się przez nie przejść, gdy zaczyna się programować. Zanim przejdziemy do dalszej części poświęćmy chwilę na zastanowienie się nad tym, czego się nauczyliśmy i dokąd zmierzamy. W tych 10 częściach skupiliśmy się całkowicie na podstawach programowania.

• Dane w postaci zmiennych i tablic.

• Control Flow - w formie instrukcji warunkowych i pętli.

• Organizacja - w postaci funkcji i obiektów.

Pojęcia te nie są unikalne dla przetwarzania i przenoszą użytkownika do wszystkich języków programowania i środowisk, takich jak C ++, Actionscript (jak we Flashu) i programowanie po stronie serwera języki takie jak PHP. Składnia może się zmienić, ale podstawowe pojęcia nie. Począwszy od części 13 książka skupi się na niektórych zaawansowanych koncepcjach dostępnych w przetwarzaniu, takich jak trójwymiarowe tłumaczenie i obrót, przetwarzanie obrazu i przechwytywanie wideo, praca w sieci i dźwięk. Chociaż pojęcia te z pewnością nie są unikalne dla przetwarzania, szczegóły ich implementacji będą bardziej szczegółowe dla środowiska, które wybraliśmy. Zanim przejdziemy do tych zaawansowanych tematów, szybko przyjrzymy się podstawowym strategiom naprawiania błędów w kodzie (część 11: Debugowanie) oraz sposobom korzystania z bibliotek przetwarzania (część 12). Wiele z tych zaawansowanych tematów wymaga importowania bibliotek dołączonych do przetwarzania, a także bibliotek tworzonych dla tej książki lub stron trzecich. Jedną z mocnych stron przetwarzania jest jego zdolność do łatwego rozszerzania o biblioteki. Zobaczymy kilka wskazówek, jak tworzyć własne biblioteki w końcowej części tego tekstu.

Projekt Piątej Lekcji

Krok 1. Opracuj pomysł na projekt, który można utworzyć za pomocą przetwarzania przy użyciu prostego rysunku kształtu i podstaw programowania. Jeśli czujesz, że utknąłeś, spróbuj stworzyć grę taką jak Pong lub Kółko i krzyżyk.

Krok 2. Postępuj zgodnie ze strategią przedstawioną w tym rozdziale i podziel ją na mniejsze części, stosując algorytm dla każdego z osobna. Upewnij się, że używasz programowania obiektowego dla każdej części.

Krok 3. Połącz mniejsze części w jednym programie. Zapomniałeś jakichś elementów lub funkcji?

Użyj przestrzeni podanej poniżej, aby szkicować projekty, notatki i pseudokod dla projektu.

XI Debugowanie

Błędy się zdarzają.

Pięć minut temu twój kod działał doskonale i przysięgasz, wszystko co zrobiłeś to zmienić kolor jakiegoś obiektu! Ale teraz, gdy statek kosmiczny uderza w asteroidę, nie kręci się już. Ale to całkowicie wirowało pięć minut temu! A twój przyjaciel zgadza się: „Tak, widziałem, jak się kręci. To było super. ”Funkcja rotate() jest dostępna. Co się stało? To powinno działać. To w ogóle nie ma sensu! Komputer jest prawdopodobnie złamany. Tak. Tak . To z pewnością wina komputera. Bez względu na to, ile czasu poświęcisz na naukę informatyki, czytanie książek programowych lub odtwarzanie nagrań dźwiękowych kodu podczas snu, mając nadzieję, że w ten sposób nasiąknie, nie ma sposobu, aby uniknąć i utknąć na błędzie. To może być naprawdę frustrujące. Błąd to każda wada programu. Czasami jest oczywiste, że masz błąd; Twój szkic zostanie zamknięty (lub w ogóle nie zostanie uruchomiony) i wyświetli błąd w konsoli komunikatów. Te typy błędów mogą być spowodowane prostymi literami, zmiennymi, które nigdy nie zostały zainicjowane, szukaniem elementu w tablicy, która nie istnieje, i tak dalej. Błędy mogą być również bardziej złowrogie i tajemnicze. Jeśli szkic przetwarzania nie działa tak, jak ty chciałbyć, masz błąd. W takim przypadku szkic może zostać uruchomiony bez błędów w konsoli komunikatów. Znalezienie tego typu błędu jest trudniejsze, ponieważ niekoniecznie będzie tak oczywiste od czego zacząć szukać w kodzie. W tej części omówimy kilka podstawowych strategii naprawiania błędów („debugowania”) w przetwarzaniu.

11.1 Porada 1: Zrób sobie przerwę.

Poważnie. Odejdź od komputera. Sen. Idź biegać. Zjedz pomarańczę. Zagraj w scrabble. Zrób coś innego niż praca nad kodem. Nie mogę ci powiedzieć, ile razy nie patrzyłem na mój kod przez wiele godzin, nie mogąc go odtworzyć, aby obudzić się następnego ranka i rozwiązać problem w ciągu pięciu minut.

11.2 Porada 2: Weź innego człowieka.

Porozmawiaj o problemie z przyjacielem. Proces pokazywania kodu innym programistom (lub nawet programistom) i przechodzenia przez głośną logikę często ujawnia błąd. W wielu przypadkach to jest czymś oczywistym, czego nie widziałeś, ponieważ dobrze znasz swój kod. Proces wyjaśniania tego komuś innemu zmusza jednak do wolniejszego przechodzenia przez kod. Jeśli nie masz znajomego w pobliżu, możesz to zrobić na głos. Tak, będziesz wyglądał głupio, ale to pomoże.

11.3 Porada 3: Uprość

Upraszczać. Upraszczać! UPRASZCZAĆ!

W części 10 skupiliśmy się na procesie przyrostowego rozwoju. Im bardziej rozwijasz swoje projekty krok po kroku, w małych, łatwych do zarządzania elementach, tym mniej błędów i błędów będziesz mieć. Oczywiście, nie ma sposobu na całkowite uniknięcie problemów, więc kiedy się pojawią, filozofię rozwoju przyrostowego można również zastosować do debugowania. Zamiast budować kod po kawałku, debugowanie polega na oddzieleniu kodu po kawałku. Jednym ze sposobów osiągnięcia tego jest skomentowanie dużych fragmentów kodu w celu wyizolowania określonej sekcji. Poniżej znajduje się główna karta przykładowego szkicu. Szkic zawiera tablicę obiektów Snake, obiekt Button i obiekt Apple. (Kod dla klas nie jest uwzględniony.) Załóżmy, że wszystko w szkicu działa poprawnie, z wyjątkiem tego, że Apple jest niewidoczne. Aby debugować problem, wszystko jest komentowane, z wyjątkiem kilku wierszy kodu, które dotyczą bezpośrednio inicjowania i wyświetlania obiektu Apple.

```
// Snake[] snakes = new Snake[100];

// Button button;
```

```
Apple apple;

void setup() {  // Tylko kod tworzący obiekt Apple i wyświetlający
                //obiekt nie jest skomentowany. W ten sposób
                //możemy być pewni, że żaden inny kod nie jest
                //przyczyną problemu.

size(200,200);

apple = new Apple();

/*for (int i = 0; i < snakes.length; i ++ ) {

snakes[i] = new Snake();

}

button = new Button(10,10,100,50);

smooth();*/

}

void draw() {

background(0);

apple.display();

// apple.move();

}

/*void mousePressed() {

button.click(mouseX,mouseY);

} */

/*for (int i = 0; i < snakes. length; i + + ) {

snakes[i].display();        // Duże bloki kodu można komentować między / * i * /

                            // / * Wszystko to jest komentowane * /

snakes[i].slither();

snakes[i].eat(apple);

}

if (button.pressed()) {

applet.restart();

} */

}

/*void mousePressed() {
```

```
button.click(mouseX,mouseY);

} */
```

Po skomentowaniu całego kodu istnieją dwa możliwe wyniki. Albo jabłko nadal się nie pojawia, albo tak. W pierwszym przypadku problem jest zdecydowanie spowodowany samym jabłkiem, a następnym krokiem byłoby zbadanie wnętrza funkcji display () i poszukiwanie błędu. Jeśli pojawi się jabłko, problem jest spowodowany jedną z pozostałych linii kodu. Być może funkcja move () wysyła ekran Apple off, abyśmy go nie widzieli. A może obiekty Snake'a zakrywają to przypadkiem. Aby to wyjaśnić, polecam odkładanie wierszy kodu pojedynczo. Za każdym razem, gdy dodajesz linię kodu, uruchom szkic i zobacz, czy jabłko zniknie. Jak tylko to zrobisz, znalazłeś sprawcę i możesz wykorzenić przyczynę. Posiadanie powyższego szkicu obiektowego (z wieloma klasami) może naprawdę pomóc w procesie debugowania. Inną taktyką, którą możesz spróbować, jest stworzenie nowego szkicu i użycie jednej z klas, testowanie jego podstawowych funkcji. Innymi słowy, nie martw się o naprawienie całego programu. Najpierw utwórz nowy szkic, który robi tylko jedną rzecz z odpowiednią klasą (lub klasami) i powiela błąd. Powiedzmy, że zamiast jabłka węże się nie zachowują prawidłowo. Aby uprościć i znaleźć błąd, możemy stworzyć szkic, który używa tylko jednego węża (zamiast tablicy) bez jabłka lub przycisku. Bez dzwonków i gwizdów kod będzie znacznie łatwiejszy.

```
Snake snake;

void setup() {

size(200,200);

snake = new Snake();

}

void draw() {

background(0);          // Ponieważ ta wersja nie zawiera obiektu Apple, nie możemy użyć tej linii
                        //kodu. W ramach procesu debugowania możemy jednak stopniowo dodawać
                        //jabłko i odkomentować ten wiersz

snakes.display();

snakes.slither();

//snakes.eat(apple);

}
```

Chociaż nie przyjrzeliśmy się jeszcze przykładom, które dotyczą urządzeń zewnętrznych (będziemy w wielu z poniższych części), uproszczenie szkicu może również obejmować wyłączenie połączeń z tymi urządzeniami, takich jak kamera, mikrofon lub połączenie sieciowe i zastąpienie ich z informacjami „obojętnymi". Na przykład o wiele łatwiej jest znaleźć problem z analizą obrazu, jeśli po prostu załadujesz plik JPG, zamiast korzystać ze źródła wideo na żywo. Lub załaduj lokalny plik tekstowy zamiast połączyć się z kanałem URL XML. Jeśli problem zniknie, możesz powiedzieć definitywnie: „Aha, prawdopodobnie serwer internetowy jest wyłączony" lub „Mój aparat musi być uszkodzony. „Jeśli nie, możesz zanurzyć się w swoim kodzie, wiedząc, że problem istnieje. Jeśli obawiasz się pogorszenia problemu poprzez usunięcie fragmentów kodu, po prostu wykonaj kopię swojego szkicu przed rozpoczęciem usuwania funkcji.

<u>11.4 Porada 4: println () jest twoim przyjacielem.</u>

Korzystanie z okna komunikatu do wyświetlania wartości zmiennych może być naprawdę pomocne. Jeśli na ekranie brakuje obiektu i chcesz wiedzieć dlaczego, możesz wydrukować wartości jego zmiennych lokalizacji. Może to wyglądać mniej więcej tak:

println("x: " + thing.x + " y: " + thing.y);

Powiedzmy, że wynik:

x: 9000000 y: - 900000

x: 9000116 y: - 901843

x: 9000184 y: - 902235

x: 9000299 y: - 903720

x: 9000682 y: - 904903

Jest całkiem oczywiste, że te wartości nie są rozsądnymi współrzędnymi pikseli. Więc coś byłoby wyłączone w sposobie, w jaki obiekt oblicza swoją lokalizację (x, y). Jeśli jednak wartości byłyby całkowicie rozsądne, to ruszyłbyś dalej. Może problem dotyczy koloru?

println(" brightness: " + brightness(thing.col) + " alpha: " + alpha(thing.col));

W wyniku:

brightness: 150.0 alpha: 0.0

Jeśli wartość alfa koloru obiektu wynosi zero, to wyjaśniałoby, dlaczego nie możesz go zobaczyć! Powinniśmy poświęcić chwilę na przypomnienie Tip # 3: Uprość. Jest to proces drukowania zmiennych wartości będzie dużo bardziej efektywne, jeśli robimy to w szkicu, który dotyczy tylko obiektu Thing. W ten sposób możemy być pewni, że nie jest to kolejna klasa, która, powiedzmy, przewija się przez przypadek. Być może zauważyłeś, że powyższe instrukcje print łączą rzeczywisty tekst ze zmiennymi. Zazwyczaj jest to dobry pomysł, aby to zrobić. Poniższy wiersz kodu wypisuje tylko wartość x bez żadnego wyjaśnienia.

println (x);

To może być mylące, aby śledzić w oknie wiadomości, zwłaszcza jeśli drukujesz różne wartości w różnych częściach kodu. Skąd wiesz, co to jest x, a co y? Jeśli umieścisz własne notatki w println (), nie może być żadnego zamieszania:

println(" The x value of the thing I’m looking for is: " + x);

Ponadto println () może być użyty do wskazania, czy dana część kodu została osiągnięta. Na przykład, jeśli w naszym przykładzie „odbijającej się piłki” piłka nigdy nie odbija się od prawej strony okna? Problemem może być (a) niewłaściwe określenie, kiedy trafi ono w krawędź, lub (b) robisz niewłaściwą rzecz, gdy uderza w krawędź. Aby wiedzieć, czy kod poprawnie wykrywa, kiedy trafi on w krawędź, możesz napisać:

if (x > width) {

println(" X is greater than width. This code is happening now! ");

xspeed * = – 1;

}

Jeśli uruchomisz szkic i nigdy nie zobaczysz wydrukowanego komunikatu, to prawdopodobnie coś zaskoczy cię wyrażeniem boolowskim. Wprawdzie println () nie jest doskonałym narzędziem do debugowania. Śledzenie wielu informacji za pomocą okna wiadomości może być trudne. Może również znacznie spowolnić szkic (w zależności od tego, ile drukujesz). Bardziej zaawansowane środowiska programistyczne zazwyczaj oferują narzędzia do debugowania, które pozwalają śledzić określone zmienne, wstrzymywać program, przesuwać wiersz po wierszu w kodzie i tak dalej. Jest to jeden z kompromisów, które wykorzystujemy podczas przetwarzania. Jest znacznie prostszy w użyciu, ale nie ma wszystkich zaawansowanych funkcji. Wciąż jednak, jeśli chodzi o debugowanie, trochę snu, trochę zdrowego rozsądku i println () może Cię całkiem nieźle zaskoczyć

XII. Biblioteki

12.1 Biblioteki

Za każdym razem, gdy wywołujemy funkcję przetwarzania, taką jak line (), background (), stroke () itd., Wywołujemy funkcję, o której dowiedzieliśmy się ze strony referencyjnej Przetwarzanie (a może nawet z tej książki!). Ta strona odniesienia jest listą wszystkich dostępnych funkcji w podstawowej bibliotece przetwarzania. W informatyce biblioteka odnosi się do zbioru kodu „pomocnika". Biblioteka może składać się z funkcji, zmiennych i obiektów. Większość rzeczy, które robimy, jest możliwa dzięki podstawowej bibliotece przetwarzania. W większości języków programowania wymagane jest określenie, które biblioteki mają być używane w górnej części kodu. To mówi kompilatorowi, gdzie szukać rzeczy, aby przetłumaczyć kod źródłowy na kod maszynowy. Jeśli miałbyś zbadać pliki znajdujące się w katalogu aplikacji Processing, znalazłbyś plik o nazwie core.jar wewnątrz biblioteki folderów. Ten plik jar zawiera skompilowany kod dla niemal wszystkiego, co robimy w przetwarzaniu. Ponieważ jest ono używane w każdym programie, przetwarzanie zakłada, że powinno zostać zaimportowane i nie wymaga jawnego pisania instrukcji importu. Gdyby tak nie było, w górnej części każdego pojedynczego szkicu znajdowałby się następujący wiersz kodu:

import processing.core.*;

„Import" wskazuje, że będziemy korzystać z biblioteki, a biblioteka, której będziemy używać, „przetwarza. rdzeń. „„ * "Jest symbolem wieloznacznym, co oznacza, że chcielibyśmy uzyskać dostęp do wszystkiego w bibliotece. Nazewnictwo biblioteki przy użyciu składni kropki (przetwarzanie rdzenia kropki) ma związek z tym, jak zbiory klas są zorganizowane w „pakiety" w języku programowania Java. Ponieważ proces przetwarzania i programowania staje się coraz bardziej komfortowy, jest to prawdopodobnie temat, który chcesz zbadać dalej. Na razie musimy tylko wiedzieć, że „processing.core" to nazwa biblioteki. Podczas gdy biblioteka podstawowa obejmuje wszystkie podstawy, w przypadku innych bardziej zaawansowanych funkcji trzeba będzie zaimportować określone biblioteki, które nie są zakładane. Nasze pierwsze spotkanie z tym programem nastąpi w Części 14, gdzie w celu skorzystania z mechanizmu renderującego OpenGL wymagana będzie biblioteka OpenGL:

import processing.opengl. *;

Wiele z poniższych części będzie wymagało wyraźnego użycia zewnętrznych bibliotek przetwarzania, takich jak wideo, sieć, serial i tak dalej. Dokumentację tych bibliotek można znaleźć na stronie internetowej Processing pod adresem http://www.processing.org/reference/libraries/. Znajdziesz tam listę bibliotek dostarczanych wraz z przetwarzaniem, a także linki do bibliotek stron trzecich dostępnych do pobrania w Internecie.

12.2 Wbudowane biblioteki

Korzystanie z wbudowanej biblioteki jest łatwe, ponieważ nie wymaga instalacji. Te biblioteki są dostarczane z aplikacją do przetwarzania. Lista wbudowanych bibliotek (pełna lista dostępna pod powyższym adresem URL) nie jest zbyt długa i wszystkie oprócz kilku są omówione w tej książce, jak podano poniżej.

• Wideo - aby przechwytywać obrazy z aparatu, odtwarzać pliki filmowe i nagrywać pliki filmowe.

• Szeregowy - Do wysyłania danych między przetwarzaniem a urządzeniem zewnętrznym za pośrednictwem komunikacji szeregowej.

• OpenGL - do renderowania szkicu z akceleracją grafiki.

• Sieć - do tworzenia szkiców klienta i serwera, które mogą komunikować się przez Internet.

• PDF - do tworzenia plików PDF o wysokiej rozdzielczości z grafiką generowaną podczas przetwarzania.

• XML - do importowania danych z dokumentów XML.

Przykłady specjalnie dostosowane do korzystania z powyższych bibliotek znajdują się w wymienionych rozdziałach. Strona internetowa Processing ma również doskonałą dokumentację dla tych bibliotek (znajduje się na stronie „biblioteki"). Jedyną potrzebną ogólną wiedzą na temat przetwarzania bibliotek wbudowanych jest to, że należy dołączyć odpowiednią instrukcję importu na górze programu. Ta instrukcja zostanie automatycznie dodana do szkicu, jeśli wybierzesz SKETCH → IMPORT LIBRARY. Lub możesz po prostu wpisać kod ręcznie (za pomocą opcji menu importowania biblioteki nie robi nic innego, jak tylko dodać tekst do instrukcji importu).

import processing.video. *;

import processing.serial. *;

import processing.opengl. *;

import processing.net. *;

import processing.pdf. *;

import processing.xml. *;

12.3 Biblioteki współdzielone

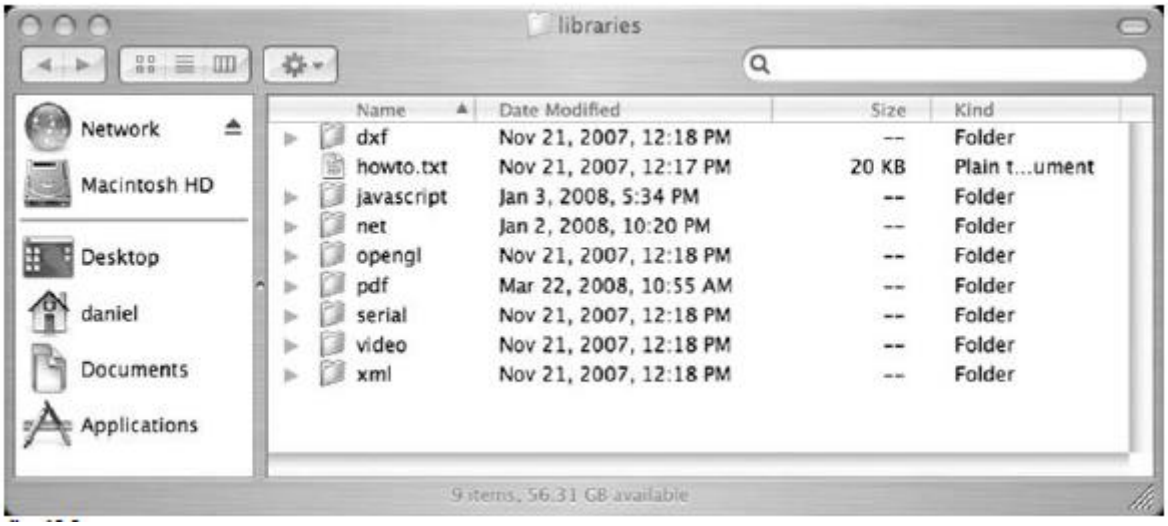
Świat stron trzecich (nazywanych również „wkładami") bibliotek do przetwarzania przypomina dziki zachód. Począwszy od napisania tej książki, istnieje 47 bibliotek, które mają możliwości od generowania dźwięku i analiza, do sniffowania pakietów, do symulacji fizycznych, do kontrolek GUI. Kilka z tych bibliotek zostanie zaprezentowanych w pozostałej części. Tutaj przyjrzymy się procesowi pobierania i instalowania biblioteki współdzielonej. Pierwszą rzeczą, którą musisz zrobić, to dowiedzieć się, gdzie zainstalowałeś Processing Na komputerze Mac aplikację najprawdopodobniej można znaleźć w katalogu „Applications" na komputerze PC, „Program Files. "Założymy to założenie, ale nie bójmy się, jeśli zainstalujesz je gdzieś indziej (Processing można zainstalować w dowolnym katalogu i będzie działać tylko po prostu), po prostu zastąp poniższą ścieżkę własną ścieżką! Po ustaleniu, gdzie zainstalowano Przetwarzanie, zajrzyj do folderu Processing

/ Aplikacje / Przetwarzanie 0135 /

lub

c: / Program Files / Processing 0135 /

W katalogu libraries znajdziesz foldery dla każdej wbudowanej biblioteki wraz z plikiem „howto.txt”, który zawiera wskazówki i instrukcje dotyczące tworzenia własnej biblioteki



Katalog libraries jest miejscem, w którym będziesz instalował biblioteki. Pierwsze użycie biblioteki innej firmy można znaleźć w części 18. Biblioteka „simpleML” przeznaczona do prostego pobierania danych HTML i XML jest dostępna do pobrania na stronie internetowej : http://www.learningprocessing.com/libraries. Jeśli chcesz rozpocząć skok do Części 18, pobierz plik simpleML.zip i postępuj zgodnie z poniższymi instrukcjami, i zilustrowany na Rysunku

Proces dla instalacja innych współtworzonych bibliotek będzie identyczna, z wyjątkiem nazw plików.

Krok 1. Rozpakuj plik ZIP. Zazwyczaj można to osiągnąć, klikając dwukrotnie plik lub dowolną aplikację dekompresyjną, taką jak Winzip na komputerze PC.

Krok 2. Skopiuj wyodrębnione pliki do folderu Biblioteki przetwarzania. Większość pobieranych bibliotek automatycznie rozpakuje się z odpowiednią strukturą katalogów. Pełna struktura katalogów powinna wyglądać następująco:

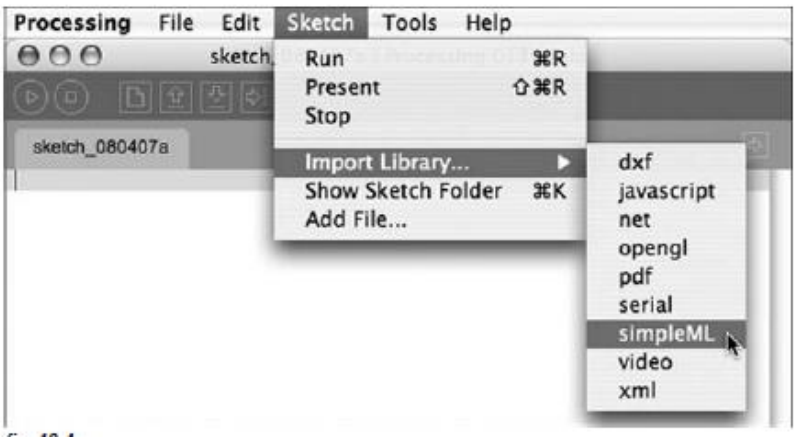
/ Processing 0135 / libraries / simpleML / library / simpleML.jar

Bardziej ogólnie:

/ Processing0135 / libraries / libraryName / library / libraryName.jar

Niektóre biblioteki mogą zawierać dodatkowe pliki w folderze „library", a także kod źródłowy (który jest zwykle przechowywany w jednym katalogu w folderze „libraryName"). Jeśli biblioteka nie rozpakuje się automatycznie za pomocą powyższej struktury katalogów, możesz ręcznie utworzyć te foldery (za pomocą programu lub eksploratora) i umieścić plik biblioteki libraryName.jar we właściwej lokalizacji.

Krok 3. Uruchom ponownie przetwarzanie. Jeśli Przetwarzanie było uruchomione podczas wykonywania kroku 2, konieczne będzie zakończenie przetwarzania i ponowne uruchomienie, aby biblioteka została rozpoznana. Po ponownym uruchomieniu jeśli wszystko poszło zgodnie z planem, biblioteka pojawi się w opcji „Szkic → Importuj bibliotekę" pokazanej na rysunku



Nowo zainstalowana biblioteka pojawi się teraz na liście! Co zrobić po zainstalowaniu biblioteki naprawdę zależy od zainstalowanej biblioteki.

XIII. Matematyka

13.1 Matematyka i programowanie

Czy kiedykolwiek zacząłeś czuć pot spływający na czoło w momencie, gdy twój nauczyciel wezwał cię do tablicy, abyś napisał rozwiązanie ostatniego zadania algebry? Czy samo wspomnienie słowa „rachunek" powoduje drżenie w kończynach? Zrelaksuj się, nie musisz się bać. Nie ma się czego obawiać, ale strach przed samą matematyką. Być może na początku czytania tej książki obawiałeś się programowania komputerowego. Z pewnością mam nadzieję, że do tej pory wszelkie przerażające wrażenia związane z kodem zostały zastąpione uczuciem spokoju. Rozdział ten ma na celu przyjęcie swobodnego i przyjaznego podejścia do kilku przydatnych tematów z matematyki, które pomogą nam w rozwoju szkiców przetwarzania. Wiesz, od zawsze używamy matematyki. Na przykład prawdopodobnie mieliśmy wyrażenie algebraiczne na prawie każdej pojedynczej stronie od czasu uczenia się zmiennych.

```
x = x + 1;
```

A ostatnio w części 10 przetestowaliśmy przecięcie za pomocą twierdzenia Pitagorasa.

```
float d = dist (x1, x2, y1, y2);
```

To tylko kilka przykładów, które widzieliśmy do tej pory, a kiedy stajesz się coraz bardziej zaawansowany, możesz nawet znaleźć się online, późno w nocy, „Googleus Spiral Inverse Curve". "Na razie zaczniemy od wyboru przydatnych tematów matematycznych.

13.2 Moduły

Zaczynamy od omówienia operatora modulo, zapisanego jako znak procentu, w przetwarzaniu. Moduł jest bardzo prostą koncepcją (taką, której nauczyłeś się bez odwoływania się do niej po nazwie, gdy studiowałeś pierwszy podział), co jest niezwykle przydatne do utrzymania liczby w obrębie pewnej granicy (kształt na ekranie, wartość indeksu w zakresie tablica itp.) Operator modulo oblicza resztę, gdy jedna liczba jest dzielona przez inną. Działa zarówno z liczbami całkowitymi, jak i zmiennoprzecinkowymi. 20 podzielone przez 6 równa się 3 pozostałości 2. (Innymi słowy: 6 * 3 + 2 = 18 + 2 = 20.) dlatego:

20 modulo 6 równa się 2 lub 20 % 6 = 2

Oto kilka innych z kilkoma pustymi miejscami do wypełnienia.

17 podzielone przez 4 równa się 4 reszta 1 : 17% 4 = 1

3 podzielone przez 5 równa się 0 reszta 3 : 3% 5 = 3

10 podzielone przez 3,75 równe 2 reszta 2,5 : 10,0 % 3,75 = 2.5

1 00 podzielone przez 50 równa się _____ reszta _____ _____ 100% 40? _____

9,25 podzielone przez 0,5 równa się _____ reszta _____ 9,25% 0,5? _____ ____

Zauważysz, że jeśli A = B % C, A nigdy nie może być większe niż C. Reszta nigdy nie może być większa niż dzielnik.

$0 \% 3 = 0$
$1 \% 3 = 1$
$2 \% 3 = 2$
$3 \% 3 = 0$
$4 \% 3 = 1$
etc.

Dlatego modulo może być używane zawsze, gdy trzeba zmienić wartość zmiennej licznika na zero. Następujące linie kodu:

```
x = x + 1;

if (x> = limit) {

x = 0;

}
```

można zastąpić przez:

```
x = (x + 1) % limit;
```

Jest to bardzo przydatne, jeśli chcesz policzyć elementy tablicy pojedynczo, zawsze powracając do 0, gdy dojdziesz do długości tablicy.

Przykład 13-1: Modulo

```
// 4 random numbers

float[] randoms = new float[4];

int index = 0; // Which number are we using

void setup() {

size(200,200);

// Fill array with random values

for (int i = 0; i < randoms.length; i + + ) {

randoms[i] = random(0,256);

}

frameRate(1);

}

void draw() {

// Every frame we access one element of the array
```

```
background(randoms[index]);

// And then go on to the next one

index = (index + 1) % randoms.length;   // Użycie operatora modulo do przełączenia licznika z
                                        //powrotem na 0.

}
```

13.3 Losowe liczby

W części 4 wprowadzono funkcję random(), która umożliwiła losowe wypełnianie zmiennych. Generator liczb losowych przetwarzania generuje tzw. „jednolity" rozkład liczb. Na przykład, jeśli poprosimy o losową liczbę od 0 do 9, 0 pojawi się 10% czasu, 1 pojawi się 10% czasu, 2 pojawi się 10% czasu i tak dalej. Moglibyśmy napisać prosty szkic za pomocą tablicy, aby to udowodnić.

Liczby pseudolosowe

Losowe liczby, które otrzymujemy z funkcji random(), nie są naprawdę losowe i są znane jako „pseudolosowe. „Są wynikiem matematycznej funkcji symulującej losowość. Ta funkcja dawałaby z czasem wzorzec, ale ten okres czasu jest tak długi, że dla nas jest równie dobry jak czysta losowość!

Przykład 13-2: Rozkład liczb losowych

```
// An array to keep track of how often random numbers are picked.

float[] randomCounts;

void setup() {

size(200,200);

randomCounts = new float[20];

}

void draw() {

background(255);

// Pick a random number and increase the count

int index = int(random(randomCounts.length));

randomCounts[index] + + ;

// Draw a rectangle to graph results

stroke(0);

fill(175);

for (int x = 0; x < randomCounts.length; x + + ) {

rect(x*10,0,9,randomCounts[x]);

}

}
```

Za pomocą kilku sztuczek możemy zmienić sposób, w jaki używamy random (), aby uzyskać nierównomierną dystrybucję liczb losowych i generować prawdopodobieństwa wystąpienia pewnych zdarzeń. Na przykład, co by było, gdybyśmy chcieli stworzyć szkic, w którym kolor tła miałby 10% szans na bycie zielonym i 90% szans na bycie niebieskim?

13.4 Przegląd prawdopodobieństwa

Przyjrzyjmy się podstawowym zasadom prawdopodobieństwa, najpierw patrząc na prawdopodobieństwo pojedynczego zdarzenia, czyli prawdopodobieństwo wystąpienia czegoś. Biorąc pod uwagę system z pewną liczbą możliwych wyników, prawdopodobieństwo wystąpienia dowolnego zdarzenia jest liczbą wyników, które kwalifikują się jako to zdarzenie podzielone przez całkowitą liczbę możliwych wyników . Najprostszym przykładem jest rzut monetą. Istnieją dwa możliwe wyniki (głowy lub ogony). Jest tylko jeden sposób odwracania głów, dlatego prawdopodobieństwo głów jest jeden podzielony przez dwa, czyli 1/2 lub 50%. Rozważ talię 52 kart. Prawdopodobieństwo wyciągnięcia asa z tej talii wynosi: liczba asów / liczba kart = 4/52 = 0,077 = ~ 8% Prawdopodobieństwo losowania diamentu wynosi: liczba diamentów / liczba kart = 13/52 = 0,25 = 25% Możemy również obliczyć prawdopodobieństwo wystąpienia wielu zdarzeń w sekwencji jako iloczyn indywidualnych prawdopodobieństw każdego zdarzenia. Prawdopodobieństwo wyrzucenia monet trzy razy z rzędu to:

(1/2) * (1/2) * (1/2) = 1/8 (lub 0,125).

Innymi słowy, moneta wyląduje trzy razy pod rząd, jeden na osiem razy (każdy „czas" to trzy rzuty).

Ćwiczenie 13-1: Jakie jest prawdopodobieństwo dobrania dwóch asów z rzędu z talii kart?

13.5 Prawdopodobieństwo zdarzenia w kodzie

Jest kilka różnych technik używania funkcji random() z prawdopodobieństwem w kodzie. Na przykład, jeśli wypełnimy tablicę z wyborem liczb (niektóre powtórzą się), możemy losowo wybrać z tej tablicy i wygenerować zdarzenia na podstawie tego, co wybieramy.

```
int[] stuff = new int[5];

stuff[0] = 1;

stuff[1] = 1;

stuff[2] = 2;

stuff[3] = 3;

stuff[4] = 3;

int index = int(random(stuff.length)); // Wybieranie losowego elementu z tablicy.

if (stuff[index] == 1) {

// do something

}
```

Jeśli uruchomisz ten kod, istnieje 40% szans na wybranie wartości 1, 20% szansy na wybranie wartości 2 i 40% szansy na wybranie wartości 3. Inną strategią jest poproszenie o liczbę losową (dla uproszczenia, rozważamy losowe wartości punktów flotacji między 0 a 1) i tylko pozwalamy, aby zdarzenie miało miejsce, jeśli losowa liczba, którą wybieramy, mieści się w pewnym zakresie. Na przykład:

```
float prob = 0.10; // A probability of 10%

float r = random(l); // A random floating point value between 0 and 1

if (r < prob) { // If our random is less than .1

/*INSTIGATE THE EVENT HERE*/  // Ten kod zostanie wykonany tylko 10% czasu.

}
```

Ta sama technika może być również stosowana do wielu wyników.

Wynik A - 60% | Wynik B - 10% | Wynik C - 30%

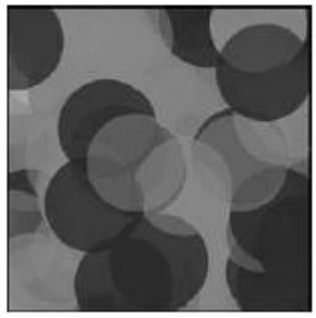
Aby zaimplementować to w kodzie, wybieramy jedną losową kupę i sprawdzamy, gdzie ona spada.

• Między 0,00 a 0,60 (10%) → wynik A.

• Między 0,60 a 0,70 (60%) → wynik B.

• Między 0,70 a 1,00 (30%) → wynik C.

Przykład 13-3 rysuje okrąg o trzech różnych kolorach, każdy z powyższym prawdopodobieństwem (czerwony: 60%, zielony: 10%, niebieski: 30%). Przykład jest pokazany na rysunku



Przykład 13-3: Prawdopodobieństwa

```
void setup() {

size(200,200);

background(255);

smooth();

noStroke();

}

void draw() {

// Probabilities for 3 different cases

// These need to add up to 100%!

float red_prob = 0.60; // 60% chance of red color

float green_prob = 0.10; // 10% chance of green color

float blue_prob = 0.30; // 30% chance of blue color
```

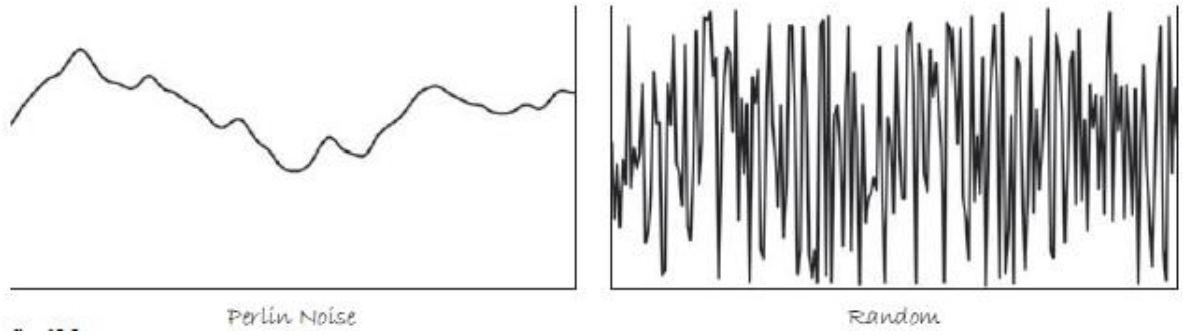
```
float num = random(1); // pick a random number between 0 and 1

// If random number is less than .6

if (num < red_prob) {

fill(255,53,2,150);

// If random number is between .6 and .7

} else if (num < green_prob + red_prob) {

fill(156,255,28,150);

// All other cases (i.e. between .7 and 1.0)

} else {

fill(10,52,178,150);

}

ellipse(random(width),random(height),64,64);

}
```

Ćwiczenie 13-2: Wypełnij puste miejsca w poniższym kodzie, aby okrąg miał 10% szansy na awans, 20% szansy na przesunięcie w dół i 70% szansy na nic.

```
float y = 100;

void setup() {

size(200,200);

smooth();

}

void draw() {

background(0);

float r = random(1);

_____

_____

_____

_____

_____

_____

ellipse(width/2,y,16,16);

}
```

13.6 Szum Perlina

Jedną z cech dobrego generatora liczb losowych jest to, że produkowane liczby wydają się nie mieć związku. Jeśli nie wykazują dostrzegalnego wzoru, są uważane za losowe. W programowaniu zachowań, które mają organiczną, niemal życiową jakość, trochę przypadkowości jest dobrą rzeczą. Nie chcemy jednak zbyt wielu losowości. Jest to podejście zastosowane przez Kena Perlina, który na początku lat 80. opracował funkcję „Szum Perlina", która tworzy naturalnie uporządkowaną (tzn. „Gładką") sekwencję liczb pseudolosowych. Pierwotnie został zaprojektowany do tworzenia tekstur proceduralnych, za które Ken Perlin zdobył Oscara za osiągnięcia techniczne. Szum Perlina może być używany do generowania różnych interesujących efektów, takich jak chmury, krajobrazy, tekstury marmuru i tak dalej. Rysunek poniższy przedstawia dwa wykresy, - wykres szumu Perlina w czasie (oś x reprezentuje czas; zwróć uwagę, jak krzywa jest gładka) w porównaniu z wykresem czystych liczb losowych w czasie. (Odwiedź stronę internetową tej książki, aby uzyskać kod, który wygenerował te wykresy).



Szczegóły hałasu

Jeśli odwiedzisz referencję szumu Processing.org, zobaczysz, że szum jest obliczany dla kilku „oktaw". "Możesz zmienić liczbę oktaw i ich względną ważność, wywołując funkcję noiseDetail (). To z kolei może zmienić zachowanie funkcji szumu. Zobacz http://processing.org/reference/noiseDetail_.html. Możesz dowiedzieć się więcej o tym, jak działa hałas od samego Kena Perlina:

http://www.noisemachine.com/talk1/.

Processing ma wbudowaną implementację algorytmu szumu Perlina z funkcją szumu (). Funkcja noise () przyjmuje jeden, dwa lub trzy argumenty (odnoszące się do „przestrzeni", w której obliczany jest hałas: jeden, dwa lub trzy wymiary). Część ta będzie dotyczyła tylko hałasu jednowymiarowego. Odwiedź stronę internetową Processing, aby uzyskać więcej informacji na temat hałasu dwuwymiarowego i trójwymiarowego. Jednowymiarowy szum Perlina generuje liniową sekwencję wartości w czasie. Na przykład:

0,364, 0,363, 0,363, 0,364, 0,365

Zwróć uwagę, że liczby przesuwają się losowo w górę lub w dół, ale pozostają blisko wartości ich poprzednika. Teraz, aby uzyskać te liczby z przetwarzania, musimy zrobić dwie rzeczy: (1) wywołać funkcję noise () i (2) przekazać jako argument bieżący „czas. „Zazwyczaj zaczynamy od czasu t = 0 i dlatego wywołujemy funkcję w ten sposób: „noise(t)"

```
float t = 0.;

float noisevalue = noise(t); // Noise at time 0
```

Możemy również pobrać powyższy kod i uruchomić go w draw().

```
float t = 0.0;

void draw() {

float noisevalue = noise(t);

println(noisevalue); // Output:

                        //.28515625

                        //0.28515625

                        //0.28515625

                        //0.28515625

}
```

Powyższy kod skutkuje taką samą wartością drukowaną w kółko. Dzieje się tak, ponieważ pytamy o wynik funkcji szumu () w tym samym punkcie „czasu" -0,0 - w kółko. Jeśli zwiększymy „czas" zmienna t, jednak otrzymamy inny wynik.

```
float t = 0.0;

void draw() {

float noisevalue = noise(t);

println(noisevalue);  // Output:

                        //0.12609221

                        //0.12697512

                        //0.12972163

                        //0.13423012

                        //0.1403218}

t + = 0.01;  // Czas idzie do przodu!

}
```

Jak szybko zwiększamy t wpływa również na gładkość szumu. Spróbuj uruchomić kod kilka razy, zwiększając t o 0,01, 0,02, 0,05, 0,1, 0,0001 i tak dalej. Być może zauważyłeś, że noise () zawsze zwraca wartość zmiennoprzecinkową z zakresu od 0 do 1. Tego szczegółu nie można przeoczyć, ponieważ wpływa on na sposób używania szumu Perlina w szkicu przetwarzania. Przykład 13-4 przypisuje wynik funkcji szumu () do rozmiaru okręgu. Wartość szumu jest skalowana przez pomnożenie przez szerokość okna. Jeśli szerokość wynosi 200, a zakres noise () wynosi od 0,0 do 1,0, zakres szumu () pomnożony przez szerokość wynosi od 0,0 do 200,0. Ilustruje to poniższa tabela i przykład 13-4.

Noise Value	Multiplied by	Equals
0	200	0
0.12	200	24
0.57	200	114
0.89	200	178
1	200	200

Przykład 13-4: Szum Perlina

```
float time = 0.0;

float increment = 0.01;

void setup() {

size(200,200);

smooth(); {

void draw() {

background(255);

float n = noise(time)*width;

// With each cycle, increment the "time"

time + = increment;

// Draw the ellipse with size determined by Perlin noise

fill(0);

ellipse(width/2,height/2,n,n);

}
```

Uzyskaj wartość szumu w „czasie” i skaluj ją zgodnie z szerokością okna.

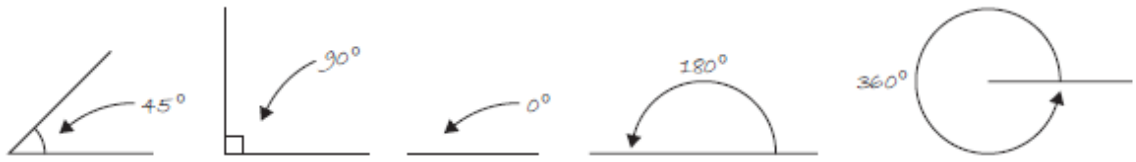
Ćwiczenie 13-3: Uzupełnij poniższy kod, który używa szumu Perlina, aby ustawić położenie okręgu. Uruchom kod. Czy koło wydaje się poruszać „naturalnie”?

```
// Noise " time " variables

float xtime = 0.0;

float ytime = 100.0;      // W tym szkicu chcemy użyć szumu dla dwóch różnych wartości. Aby wyjście
                          //funkcji szumu nie było identyczne, zaczynamy w dwóch różnych punktach w
                          //czasie.

float increment = 0.01;

void setup() {
```

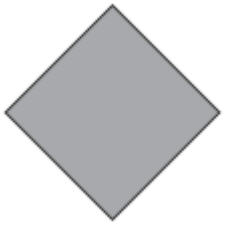
```
size(200,200);

smooth();

}

void draw() {

background(0);

float x = _____;

float y = _____;

_____;

_____;

// Draw the ellipse with size determined by Perlin noise

fill(200);

ellipse(_____,_____,_____,_____);

}
```

13.7 Kąty

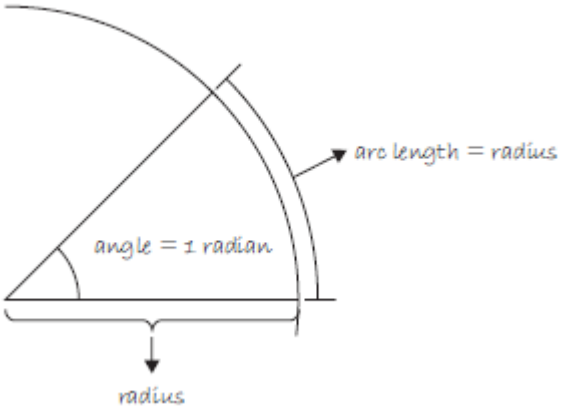
Niektóre przykłady będą wymagały podstawowej wiedzy na temat sposobu definiowania kątów w przetwarzaniu. W części 14, na przykład, będziemy musieli wiedzieć o kątach, aby czuć się komfortowo, używając funkcji rotate () do obracania i obracania obiektów. Aby przygotować się na te nadchodzące przykłady, musimy dowiedzieć się o radianach i stopniach. Prawdopodobnie znasz pojęcie kąta w stopniach. Pełny obrót wynosi od zera do 360 °. Kąt 90 ° (kąt prosty) wynosi jedną czwartą 360 °, co pokazano na rysunku jako dwie prostopadłe linie.



Jest dość intuicyjne, aby myśleć o kątach w kategoriach stopni. Dla przykładu prostokąt na rysunku



obraca się o 45 ° wokół jego środka. Processing wymaga jednak określenia kątów w radianach. Radian jest jednostką miary dla kątów określonych przez stosunek długości łuku koła do promienia tego okręgu. Jeden radian to kąt, pod którym ten stosunek jest równy jeden



Kąt 180 ° = PI radianów. Kąt 360 ° = 2 * radianów PI i 90 ° = PI / 2 radianów i tak dalej.

Formuła konwersji z stopni na radiany to:

Radiany = 2 * PI * (stopnie / 360)

Na szczęście dla nas, jeśli wolimy myśleć w stopniach, ale kodować z radianami, przetwarzanie to ułatwia. Funkcja radians() automatycznie konwertuje wartości z stopni na radiany. Ponadto stałe PI i TWO_PI są dostępne dla wygodnego dostępu do tych powszechnie używanych liczb (odpowiednik odpowiednio 180 ° i 360 °). Poniższy kod, na przykład, obróci kształty o 60 ° (obrót zostanie w pełni zbadany w następnej części).

```
float angle = radians(60);

rotate(angle);
```

PI, co to jest?

Stała matematyczna PI (lub π) jest liczbą rzeczywistą zdefiniowaną jako stosunek obwodu okręgu (odległość wokół obwodu) do jego średnicy (linia prosta przechodząca przez środek okręgu). Jest on równy około 3,14159. Jest określany jako stosunek obwodu koła (odległość wokół obwodu) do jego średnicy (linia prosta przechodząca przez środek okręgu). Jest równa około 3,14159.

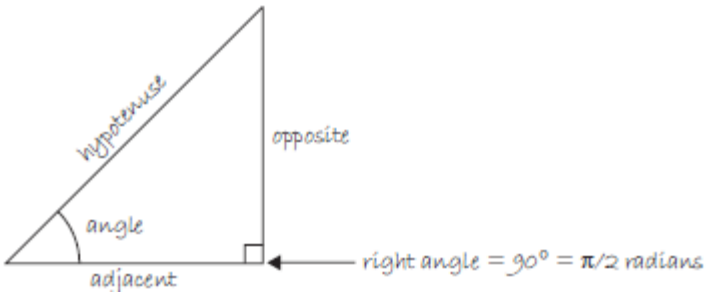
Ćwiczenie 13-4: Tancerz obraca się wokół dwóch pełnych obrotów. Ile stopni obrócił tancerz? Ile radianów?

Stopnie: _____ Radiany: _____

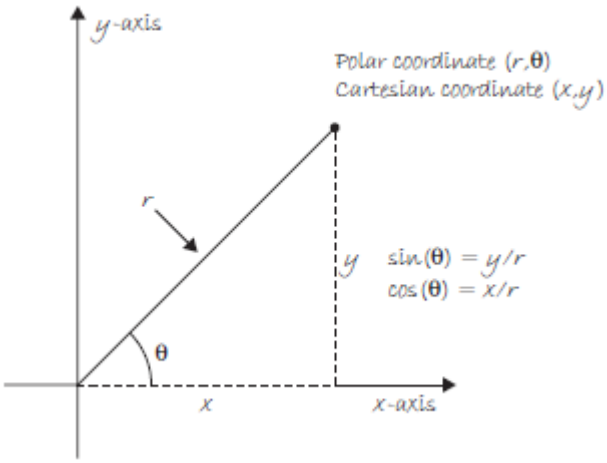
13.8 Trygonometria

Sohcahtoa. O dziwo, to pozornie bzdurne słowo, sohcahtoa, jest podstawą wielu prac związanych z grafiką komputerową. Za każdym razem, gdy musisz obliczyć kąt, określ odległość między punktami, zajmuj się okręgami, łukami, liniami itd., znajdziesz podstawowe informacje na temat trygonometrii. Trygonometria to badanie relacji między bokami i kątami trójkątów, a sohcahtoa jest urządzeniem mnemonicznym do zapamiętywania definicji funkcji trygonometrycznych, sinus, cosinus i stycznej.

- soh : sine = opposite/hypotenuse
- cah : cosine = adjacent/hypotenuse
- toa : tangent = opposite/adjacent



Za każdym razem, gdy wyświetlamy kształt w przetwarzaniu, musimy określić położenie piksela, podane jako współrzędne x i y. Współrzędne te znane są jako współrzędne kartezjańskie, nazwane na cześć francuskiego matematyka René Descartesa, który opracował idee kryjące się za przestrzenią kartezjańską. Inny użyteczny układ współrzędnych, znany jako współrzędne biegunowe, opisuje punkt w przestrzeni jako kąt obrotu wokół początku i promienia od początku. Nie możemy użyć współrzędnych biegunowych jako argumentów funkcji w Przetwarzaniu. Jednak wzory trygonometryczne pozwalają nam przekształcić te współrzędne na kartezjańskie, które następnie można wykorzystać do narysowania kształtu.



Grecka litera θ często jest używana dla określania kąta

$$\text{sine(theta)} = y/r \rightarrow y = r * \text{sine(theta)}$$
$$\text{cosine(theta)} = x/r \rightarrow x = r * \text{cosine(theta)}$$

Na przykład, jeśli r wynosi 75, a theta wynosi 45 ° (lub PI / 4 radiany), możemy obliczyć xiy w następujący sposób. Funkcje sinus i cosinus w Processing to odpowiednio sin() i cos(). Każdy z nich przyjmuje jeden argument, kąt zmiennoprzecinkowy mierzony w radianach.

```
float r = 75;

float theta = PI / 4; // We could also say: float theta = radians(45);

float x = r * cos(theta);

float y = r * sin(theta);
```

Ten typ konwersji może być przydatny w niektórych aplikacjach. Na przykład, w jaki sposób można przesunąć kształt wzdłuż ścieżki kołowej przy użyciu współrzędnych kartezjańskich? To byłoby trudne. Jednak za pomocą współrzędnych biegunowych zadanie to jest łatwe. Po prostu zwiększ kąt! Oto jak to się robi ze zmiennymi globalnymi r i theta.

Przykład 13-5: Biegunowy do kartezjański

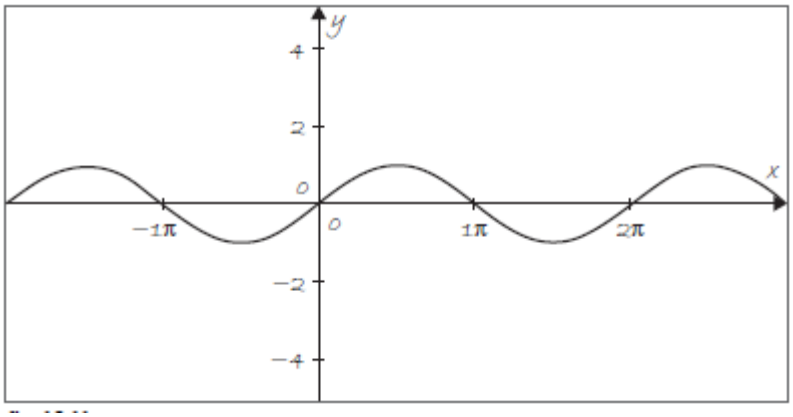
```
/ A Polar coordinate

float r = 75;

float theta = 0;

void setup() {

size(200,200);

background(255);

smooth();

}

void draw() {

// Polar to Cartesian conversion

float x = r * cos(theta);

float y = r * sin(theta);

// Draw an ellipse at x,y

noStroke();

fill(0);

ellipse(x + width/2, y + height/2, 16, 16); // Adjust for center of window

// Increment the angle

theta + = 0.01;

}
```

Współrzędne biegunowe (r, theta) są konwertowane na kartezjańskie (x, y) do użycia w funkcji ellipse ().

Ćwiczenie 13-5: Za pomocą przykładu 13-5 narysuj ścieżkę spiralną. Zacznij od środka i ruszaj na zewnątrz. Zauważ, że można to zrobić zmieniając tylko jeden wiersz kodu i dodając jeden wiersz kodu!

13.9 Oscylacja

Funkcje trygonometryczne mogą być używane do więcej niż obliczeń geometrycznych związanych z trójkątami prawymi. Spójrzmy na rysunek, wykres funkcji sinus, gdzie y = sin (x).



Zauważysz, że wyjście sinus jest gładką krzywą na przemian pomiędzy –1 a 1. Jest to typ zachowania znany jako oscylacja, okresowy ruch między dwoma punktami. Kołysanie wahadła przykład oscyluje. Możemy symulować oscylacje w szkicu przetwarzania, przypisując dane wyjściowe funkcji sinus do lokalizacji obiektu. Jest to podobne do tego, w jaki sposób używaliśmy szumu () do kontrolowania rozmiaru okręgu, tylko z sin () kontrolującym lokalizację. Zauważ, że podczas gdy noise () generuje liczbę z przedziału od 0 do 1.0, sin() podaje zakres od –1 do 1. Przykład 13-6 pokazuje kod wahadła oscylacyjnego.

Przykład 13-6: Oscylacja

```
float theta = 0.0;

void setup() {

size(200,200);

smooth();

}

void draw() {

background(255);

// Get the result of the sine function
```

// Scale so that values oscillate between 0 and width // Wyjście funkcji sin () oscyluje płynnie //między -1 i 1. Dodając 1 otrzymujemy //wartości od 0 do 2. Poprzez pomnożenie o //100 otrzymujemy wartości od 0 do 200, //które mogą być użyte jako lokalizacja x elipsy.

float x = (sin(theta) + 1) * width/2;

// With each cycle, increment theta

theta + = 0.05;

// Draw the ellipse at the value produced by sine

fill(0);

stroke(0);

line(width/2,0,x,height/2);

ellipse(x,height/2,16,16);

}

Ćwiczenie 13-6: Dołącz powyższą funkcjonalność do obiektu oscylatora. Utwórz tablicę oscylatorów, z których każdy porusza się z różnymi prędkościami wzdłuż osi xiy. Oto kod klasy Oscillator, który pomoże Ci zacząć.

class Oscillator {

float xtheta;

float ytheta;

_____

Oscillator() {

xtheta = 0;

ytheta = 0;

_____

}

void oscillate() {

_____

_____

}

void display() {

float x = _____

float y = _____

```
ellipse(x,y,16,16);

}

}
```

Ćwiczenie 13-7: Użyj funkcji sinus, aby utworzyć kształt „oddychający”, czyli taki, którego rozmiar oscyluje. Możemy również uzyskać interesujące wyniki, rysując sekwencję kształtów wzdłuż ścieżki funkcji sinus

Przykład 13-7: Fala

```
// Starting angle

float theta = 0.0;

void setup() {

size(200,200);

smooth();

}

void draw() {

background(255);

// Increment theta (try different values for " angular velocity " here)

theta + = 0.02;

noStroke();

fill(0);

float x = theta;

// A simple way to draw the wave with an ellipse at each location

for (int i = 0; i < = 20; i + + ) {

// Calculate y value based off of sine function

float y = sin(x)*height/2;

// Draw an ellipse

ellipse(i*10,y + height/2,16,16);      // Pętla for jest używana do rysowania wszystkich punktów
                                       //wzdłuż fali sinusoidalnej (przeskalowana do wymiaru piksela
                                       //okna)

// Move along x-axis

x + = 0.2;

}

}
```

Ćwiczenie 13-8: Przepisz powyższy przykład, aby użyć funkcji noise() zamiast sin()

13.10 Rekurencja

W 1975 roku Benoit Mandelbrot ukuł termin „fraktal", aby opisać podobne do siebie kształty występujące w przyrodzie. Wiele rzeczy, które spotykamy w naszym świecie fizycznym, można opisać wyidealizowanymi formami geometrycznymi - pocztówka ma kształt prostokątny, piłka do ping-ponga jest kulista i tak dalej. Jednak wielu naturalnie występujących struktur nie można opisać tak prostymi środkami. Niektóre przykłady to płatki śniegu, drzewa, linie brzegowe i góry. Fraktale dostarczają geometrii do opisywania i symulowania tych samopodobnych kształtów (przez „samopodobny" rozumiemy, niezależnie od tego, jak „powiększony" lub „powiększony", kształt ostatecznie wydaje się taki sam). Jeden proces generowania tych kształtów jest znany jako rekursja. Wiemy, że funkcja może wywołać inną funkcję. Robimy to za każdym razem, gdy wywołujemy dowolną funkcję wewnątrz funkcji draw (). Ale czy funkcja może zadzwonić sama? Czy draw () wywołuje draw ()? W rzeczywistości może (chociaż wywołanie funkcji draw () z draw () jest okropnym przykładem, ponieważ skutkowałoby nieskończoną pętlą). Funkcje, które same się nazywają, są rekurencyjne i są odpowiednie do rozwiązywania różnych typów problemów. Dzieje się tak w obliczeniach matematycznych; najczęstszym tego przykładem jest „silnia. „Silnia dowolnej liczby n, zwykle napisana jako n!, jest zdefiniowana jako:

$$n! = n * n-1 * \ldots * 3 * 2 * 1$$
$$0! = 1$$

Moglibyśmy napisać funkcję do obliczenia silni za pomocą pętli for w przetwarzaniu:

```
int factorial(int n) {

int f = 1;

for (int i = 0; i < n; i + + ) {

f = f * (i + 1);

}

return f;

}
```

Jeśli jednak przyjrzysz się bliżej, jak działa silnia, zauważysz coś interesującego. Przyjrzyjmy się 4! i 3!

4! = 4 * 3 * 2 * 1

3! = 3 * 2 * 1

dlatego. . . 4! = 4 * 3!

Możemy to opisać w bardziej ogólny sposób. Dla każdej dodatniej liczby całkowitej n:

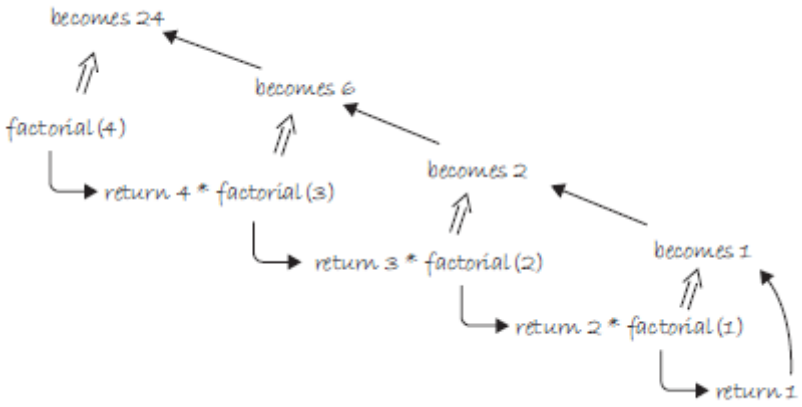
$$n! = n * (n-1)!$$
$$1! = 1$$

Silnia N jest zdefiniowana jako N razy silnia N - 1.

Definicja silni obejmuje silnię?! To jakby powiedzieć „zmęczony” jest zdefiniowany jako „uczucie, które odczuwasz, gdy jesteś zmęczony. „Jest to koncepcja samoodniesienia w funkcjach, zwana rekurencją. Możemy także użyć rekurencji, aby napisać funkcję dla silni, która wywołuje samą siebie.

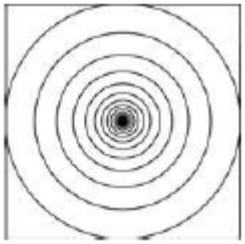
```
int factorial(int n) {

if (n == 1) {

return 1;

} else {

return n * factorial(n-1);

}

}
```

Szalone, wiem. Ale to działa. Rysunek omawia kroki, które mają miejsce, gdy wywoływana jest silnia (4)



Ta sama zasada może być stosowana do grafiki z interesującymi wynikami. Spójrz na następującą funkcję rekurencyjną. Wyniki przedstawiono na rysunku

```
void drawCircle(int x, int y, float radius) {

ellipse(x, y, radius, radius);

if(radius > 2) {

radius * = 0.75f;

drawCircle(x, y, radius);

}

}
```



Co robi drawCircle ()? Rysuje elipsę na podstawie zestawu parametrów otrzymanych jako argumenty, a następnie wywołuje się z tymi samymi parametrami (nieznacznie je dopasowując). Rezultatem jest seria okręgów narysowanych w poprzednim okręgu. Zauważ, że powyższa funkcja wywołuje się rekurencyjnie tylko wtedy, gdy promień jest większy niż dwa. To jest kluczowy punkt. Wszystkie funkcje rekurencyjne muszą mieć warunek wyjścia! To jest identyczne z iteracją. W części 6 dowiedzieliśmy się, że wszystkie pętle for i while muszą zawierać test boolowski, który ostatecznie oceni na false, kończąc w ten sposób pętlę. Bez niego program ulegnie awarii, przechwycony w nieskończonej pętli. To samo można powiedzieć o rekursji. Jeśli funkcja rekursywna wywoła siebie na zawsze i zawsze, najprawdopodobniej zostaniesz potraktowany na ładnym, zamrożonym ekranie. Powyższy przykład kół jest dość trywialny, ponieważ można go łatwo osiągnąć poprzez prostą iterację. Jednak w bardziej złożonych scenariuszach, w których metoda wywołuje się więcej niż raz, rekurencja staje się cudownie elegancka. Zmieńmy drawCircle (), aby było nieco bardziej złożone. Dla każdego wyświetlanego okręgu narysuj okrąg o połowę mniejszy po lewej i prawej stronie tego okręgu.

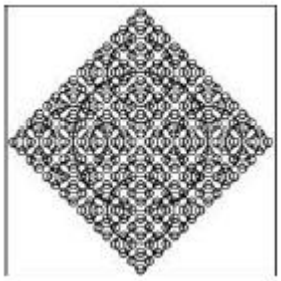
Przykład 13-8: Rekurencja

```
void setup() {
size(200,200);
smooth();
}
void draw() {
background(255);
stroke(0);
noFill();
drawCircle(width/2,height/2,100);
}
void drawCircle(float x, float y, float radius) {
ellipse(x, y, radius, radius);
if(radius > 2) {
drawCircle(x + radius/2, y, radius/2);
drawCircle(x – radius/2, y, radius/2);   // drawCircle () wywołuje się dwukrotnie, tworząc efekt
                                         //rozgałęzienia. Dla każdego okręgu rysowany jest mniejszy
                                         //okrąg po lewej i prawej stronie.
```

```
}
}
```

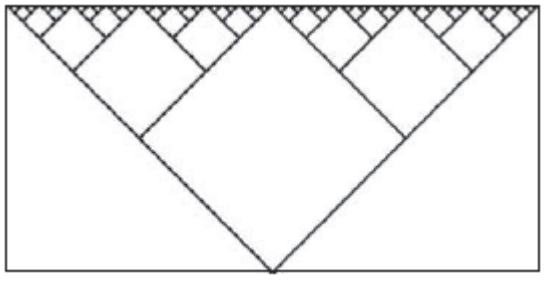
Mając nieco mniejszy kod, możemy dodać okrąg powyżej i poniżej. Wynik jest pokazany na rysunku

```
void drawCircle(float x, float y, float radius) {
ellipse(x, y, radius, radius);
if(radius > 8) {
drawCircle(x + radius/2, y, radius/2);
drawCircle(x – radius/2, y, radius/2);
drawCircle(x, y + radius/2, radius/2);
drawCircle(x, y – radius/2, radius/2);
}
}
```



Po prostu spróbuj odtworzyć ten szkic z iteracją zamiast rekurencji! Wyzywam cię!

Ćwiczenie 13-9: Uzupełnij kod, który generuje następujący wzór



```
void setup() {
size(400,200);
}
void draw() {
background(255);
stroke(0);
```

```
branch(width/2,height,100);

}

void branch(float x, float y, float h) {

_____;

_____;

if (_____) {

_____;

_____;

}

}
```

Tablica dwuwymiarowa jest tak naprawdę tylko tablicą tablic (tablica trójwymiarowa jest tablicą tablic tablic). Tusz twojego obiadu. Mógłbyś mieć jednowymiarową listę wszystkiego, co jesz: (sałata, pomidory, sos sałatkowy, stek, puree ziemniaczane, fasolka szparagowa, ciasto, lody, kawa). Albo możesz mieć dwuwymiarową listę trzech dań, każda zawierające trzy rzeczy, które jesz: (sałata, pomidory, sos sałatkowy) i (stek, tłuczone ziemniaki, fasolka szparagowa) i (ciasto, lody, kawa) W przypadku tablicy, nasz staromodny jednowymiarowy zestaw wygląda jak to:

```
int[ ] myArray = { 0,1,2,3};
```

A tablica dwuwymiarowa wygląda tak:

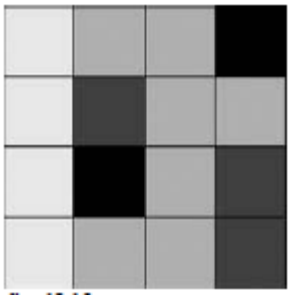
```
int[ ][ ] myArray = { { 0,1,2,3},{3,2,1,0},{3,5,6,1},{3,8,3,4} } ;
```

Dla naszych celów lepiej jest myśleć o dwuwymiarowej tablicy jako macierzy. Macierz może być traktowana jako siatka liczb, ułożonych w rzędy i kolumny, trochę jak plansza do gry w bingo. Możemy napisać dwuwymiarową tablicę w następujący sposób, aby zilustrować ten punkt:

```
int[ ][ ] myArray = { {0, 1, 2, 3 },

{ 3, 2, 1, 0 },

{ 3, 5, 6, 1 },

{ 3, 8, 3, 4 } };
```

Możemy użyć tego typu struktury danych do kodowania informacji o obrazie. Na przykład obraz w skali szarości na rysunku 13.19 może być reprezentowany przez następującą tablicę:

```
int[ ][ ] myArray = { {236, 189, 189, 0},

{ 236, 80, 189, 189 },

{ 236, 0, 189, 80},

{ 236, 189, 189, 80} };
```

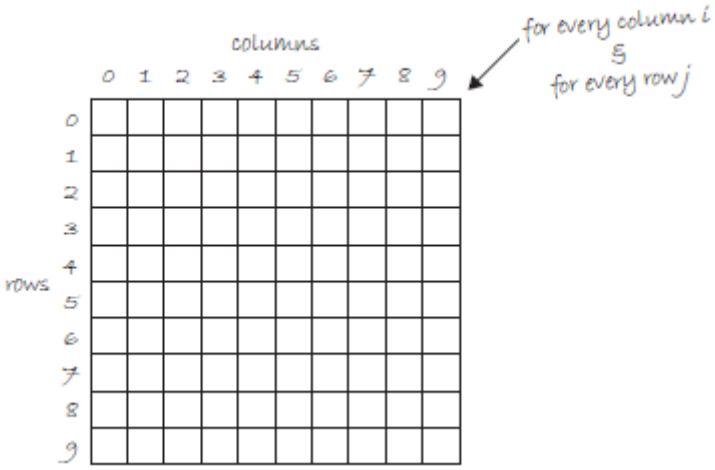


Aby przejść przez każdy element jednowymiarowej tablicy, używamy pętli for, czyli:

```
int[ ] myArray = new int[10];

for (int i = 0; i < myArray.length; i + + ) {

myArray[i] = 0;

}
```

W przypadku tablicy dwuwymiarowej, aby odwołać się do każdego elementu, musimy użyć dwóch zagnieżdżonych pętli. Daje nam to zmienną licznika dla każdej kolumny i każdego wiersza w macierzy.

```
int cols = 10;

int rows = 10;

int[ ][ ] myArray = new int[cols][rows];

for (int i = 0; i < cols; i + + ) {     // Dwie zagnieżdżone pętle pozwalają nam odwiedzać każde miejsce
                                        //w dwuwymiarowej tablicy. W każdej kolumnie i odwiedź każdy
                                        //wiersz j.

for (int j = 0; j < rows; j + + ) {

myArray[i][j] = 0;

}

}
```



Na przykład możemy napisać program wykorzystujący dwuwymiarową tablicę do narysowania obrazu w skali szarości, jak w przykładzie 13–9.

Przykład 13-9: Tablica dwuwymiarowa

```
// Set up dimensions

size(200,200);

int cols = width;

int rows = height;

// Declare 2D array

int[][] myArray = new int[cols][rows];

// Initialize 2D array values

for (int i = 0; i < cols; i + + ) {

for (int j = 0; j < rows; j + + ) {

myArray[i][j] = int(random(255));

}

}

// Draw points

for (int i = 0; i < cols; i + + ) {

for (int j = 0; j < rows; j + + ) {

stroke(myArray[i][j]);

point(i,j);

}

}
```

Tablica dwuwymiarowa może być również wykorzystywana do przechowywania obiektów, co jest szczególnie wygodne w przypadku szkiców programistycznych, które wymagają pewnego rodzaju „siatki" lub „planszy". "Przykład 13-10 wyświetla siatkę obiektów Cell przechowywanych w dwuwymiarowej tablicy. Każda komórka jest prostokątem, którego jasność oscyluje od 0-255 z funkcją sinus

```
// 2D Array of object // Dwuwymiarowa tablica może być używana do przechowywania obiektów.

Cell[][] grid;

// Number of columns and rows in the grid

int cols = 10;

int rows = 10;

void setup() {
```

```
size(200,200);
grid = new Cell[cols][rows];
for (int i = 0; i < cols; i + + ) {
for (int j = 0; j < rows; j + + ) {
// Initialize each object
grid[i][j] = new Cell(i*20,j*20,20,20,i + j);
}
}
}
void draw() {     // Zmienne licznikowe i i j są również numerami kolumn i wierszy i są używane jako
                  //argumenty dla konstruktora dla każdego obiektu w siatce
background(0);
for (int i = 0; i < cols; i + + ) {
for (int j = 0; j < rows; j + + ) {
// Oscillate and display each object
grid[i][j].oscillate();
grid[i][j].display();
}
}
}
// A Cell object          // Obiekt komórki wie o jego lokalizacji w siatce, a także o jego wielkości za
                          //pomocą zmiennych x, y, w, h.
class Cell {
float x,y; // x,y location
float w,h; // width and height
float angle; // angle for oscillating brightness
// Cell Constructor
Cell(float tempX, float tempY, float tempW, float tempH, float tempAngle) {
x = tempX;
y = tempY;
w = tempW;
h = tempH;
```

```
angle = tempAngle;
}
// Oscillation means increase angle
void oscillate() {
angle + = 0.02;
}
void display() {
stroke(255);
// Color calculated using sine wave
fill(127 + 127*sin(angle));
rect(x,y,w,h);
}
}
```

Ćwiczenie 13-10: Rozwiń początki gry Kółko i krzyżyk. Utwórz obiekt Cell, który może istnieć w jednym z dwóch stanów: O lub nic. Po kliknięciu komórki jej stan zmienia się z nic na „O”. Oto ramy, dzięki którym możesz zacząć.

```
Cell[][] board;
int cols = 3;
int rows = 3;
void setup() {
// FILL IN
}
void draw() {
background(0);
for (int i = 0; i < cols; i + + ) {
for (int j = 0; j < rows; j + + ) {
board[i][j].display();
}
}
}
void mousePressed() {
// FILL IN
```

```
}

// A Cell object

class Cell {

float x,y;

float w,h;

int state;

// Cell Constructor

Cell(float tempX, float tempY, float tempW, float tempH) {

// FILL IN

}

void click(int mx, int my) {

// FILL IN

}

void display() {

// FILL IN

}

}
```

Ćwiczenie 13-11: Jeśli jesteś zuchwały, idź dalej i ukończ grę Kółko i krzyżyk, dodając X i zmieniając kolejność graczy za pomocą kliknięć myszą.

XIV. Przesunięcia i obroty (w 3D!)

14.1 Oś Z.

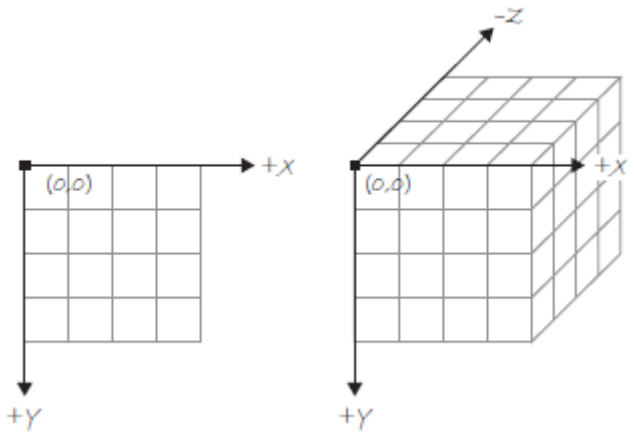
Jak widzieliśmy do tej pory, piksele w dwuwymiarowym oknie są opisane za pomocą współrzędnych kartezjańskich: punktu X (poziomego) i Y (pionowego). Te dane sięgają aż do części 1, kiedy zaczęliśmy myśleć o ekranie jako cyfrowym kawałku papieru milimetrowego. W przestrzeni trójwymiarowej trzecia oś (powszechnie określana mianem osi Z) odnosi się do głębokości dowolnego punktu. W oknie szkicu przetwarzania współrzędna wzdłuż osi Z wskazuje, jak daleko przed lub za oknem żyje piksel. Drapanie się po głowie to tutaj całkiem rozsądna odpowiedź. W końcu okno komputera jest tylko dwuwymiarowe. Nie ma pikseli pływających w powietrzu przed lub za monitorem LCD! W tej części przyjrzymy się, w jaki sposób użycie teoretycznej osi Z stworzy iluzję trójwymiarowej przestrzeni w oknie przetwarzania. W rzeczywistości możemy stworzyć trójwymiarową iluzję z tym, czego do tej pory się nauczyliśmy. Na przykład, jeśli narysujesz prostokąt na środku okna i powoli zwiększysz jego szerokość i wysokość, może się wydawać, że porusza się on ku tobie

Przykład 14-1: Rosnący prostokąt lub prostokąt poruszający się w twoim kierunku?

```
float r = 8;

void setup() {

size(200,200);

}

void draw() {

background(255);

// Display a rectangle in the middle of the screen

stroke(0);

fill(175);

rectMode(CENTER);

rect(width/2,height/2,r,r);

// Increase the rectangle size

r + + ;

}
```

Czy ten prostokąt odlatuje z ekranu komputera, który ma uderzyć w twój nos? Technicznie rzecz biorąc, oczywiście tak nie jest. Jest to po prostu prostokąt o rozmiarze rosnącym. Ale stworzyliśmy iluzję prostokąta poruszającego się w twoim kierunku. Na szczęście dla nas, jeśli wybierzemy współrzędne 3D, Processing stworzy dla nas iluzję. Podczas gdy idea trzeciego wymiaru na płaskim monitorze komputera może wydawać się wyimaginowana, jest całkiem realna dla przetwarzania. Processing wie o perspektywie i wybiera odpowiednie dwuwymiarowe piksele, aby stworzyć efekt trójwymiarowy. Powinniśmy jednak zdawać sobie sprawę z tego, że jak tylko wejdziemy w świat współrzędnych pikseli 3D, należy przekazać pewną kontrolę do renderera przetwarzania. Nie można już kontrolować dokładnych lokalizacji pikseli, tak jak w przypadku kształtów 2D, ponieważ lokalizacje XY zostaną dostosowane do uwzględnienia perspektywy 3D. Aby określić punkty w trzech wymiarach,

współrzędne są określone w kolejności, w jakiej można by oczekiwać: x, y, z. Kartezjańskie systemy 3D można opisać jako „leworęczne” lub „praworęczne”. ”Jeśli użyjesz prawej ręki, aby skierować wskaźnik w kierunku dodatnim y (w górę) i kciukiem w kierunku dodatnim x (w prawo), reszta palców będzie wskazywać kierunek dodatni. Jest leworęczny, jeśli używasz lewej ręki i robisz to samo. W przetwarzaniu system jest leworęczny, jak pokazano na rysunku



Naszym pierwszym celem jest przepisanie przykładu 14-1 z wykorzystaniem możliwości przetwarzania 3D. Załóż następujące zmienne:

int x = width/2;

int y = height/2;

int z = 0;

int r = 10;

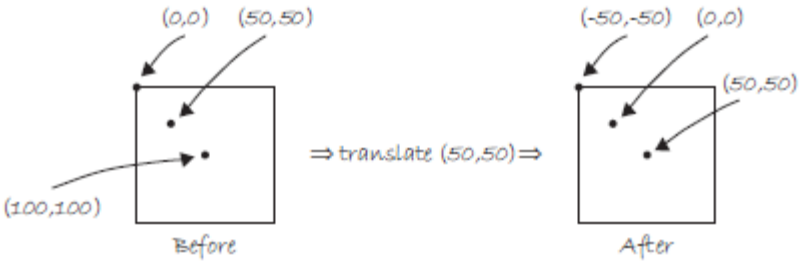
Aby określić położenie prostokąta, funkcja rect () pobiera cztery argumenty: położenie x, położenie y, szerokość i wysokość.

rect (x, y, w, h);

Naszym pierwszym instynktem może być dodanie kolejnego argumentu do funkcji rect().

```
rect (x, y, z, w, h);      // Błędny! Nie możemy używać współrzędnych (x, y, z) w funkcjach
                           //kształtowania przetwarzania, takich jak rect (), ellipse (), line () i tak dalej.
                           //Inne funkcje w przetwarzaniu mogą przyjmować trzy argumenty dla x, y, z i
                           //zobaczymy to później w części
```

Strona referencyjna przetwarzania dla rect () nie pozwala jednak na taką możliwość. Aby określić współrzędne 3D dla kształtów w świecie przetwarzania, musimy nauczyć się korzystać z nowej funkcji o nazwie translate (). Funkcja translate () nie dotyczy wyłącznie szkiców 3D, więc powróćmy do dwóch wymiarów, aby zobaczyć, jak działa. Funkcja translate () przesuwa punkt początkowy - (0,0) - zależny od poprzedniego stanu. Wiemy, że po pierwszym uruchomieniu szkicu punkt początkowy znajduje się w lewym górnym rogu okna. Gdybyśmy wywołali funkcję translate () z argumentami (50,50), wynik byłby taki, jak pokazano na rysunku



Gdzie jest początek?

„Początek" w szkicu przetwarzania to punkt (0,0) w dwóch wymiarach lub (0,0,0) w trzech wymiarach. Zawsze jest w lewym górnym rogu okna, chyba że przeniesiesz go za pomocą translate().

Możesz myśleć o tym, jak przesuwać pióro po ekranie, gdzie pióro wskazuje punkt początkowy. Ponadto początek zawsze resetuje się z powrotem do lewego górnego rogu na początku drawa (). Wszelkie wywołania translate () dotyczą tylko bieżącego cyklu za pośrednictwem pętli draw ().

Przykład 14-2: Wiele przesunięć

```
void setup() {

size(200,200);

smooth();

}

void draw() {

background(255);

stroke(0);

fill(175);

// Grab mouse coordinates, constrained to window

int mx = constrain(mouseX,0,width);

int my = constrain(mouseY,0,height);

// Translate to the mouse location

translate(mx,my);

ellipse(0,0,8,8);

// Translate 100 pixels to the right

translate(100,0);

ellipse(0,0,8,8);

// Translate 100 pixels down

translate(0,100);
```

```
ellipse(0,0,8,8);

// Translate 100 pixels left

translate(–100,0);

ellipse(0,0,8,8);

}
```

Teraz, gdy rozumiemy, jak działa translate (), możemy powrócić do pierwotnego problemu określania współrzędnych 3D. translate (), w przeciwieństwie do rect (), ellipse () i innych funkcji kształtu, może przyjąć trzeci argument dla współrzędnej Z.

```
// Translation along the z-axis

translate(0,0,50);

rectMode(CENTER);

rect(100,100,8,8);
```

Th e above code translates 50 units along the Z -axis, and then draws a rectangle at (100,100). While the

above is technically correct, when using translate( ) , it is a good habit to specify the (x , y) location as part

of the translation, that is:

```
// Translation along the z-axis II

translate(100,100,50);

rectMode(CENTER);

rect(0,0,8,8);     //Podczas używania translate () położenie prostokąta wynosi (0,0), ponieważ translate
                   //() przeniosło nas do lokalizacji prostokąta.
```

Na koniec możemy użyć zmiennej dla lokalizacji Z i animować kształt przesuwający się w naszym kierunku.

```
float z = 0; // a variable for the Z (depth) coordinate

void setup() {

size(200,200,P3D);     // Używając współrzędnych (x, y, z), musimy powiedzieć Przetwarzanie, że
                       //chcemy szkic 3D. Odbywa się to przez dodanie trzeciego argumentu „P3D”
                       //do funkcji size ().

}

void draw() {

background(0);

stroke(255);

fill(100);
```

```
// Translate to a point before displaying a shape there

translate(width/2,height/2,z);

rectMode(CENTER);

rect(0,0,8,8);

z + + ; // Increment Z (i.e. move the shape toward the viewer)

}
```

Chociaż wynik nie wygląda inaczej niż w przykładzie 14-1, jest on zupełnie inny pod względem koncepcyjnym, ponieważ otworzyliśmy drzwi do tworzenia różnorodnych efektów trójwymiarowych na ekranie za pomocą silnika 3D Processing

Ćwiczenie 14-1: Wypełnij odpowiednie funkcje translate (), aby utworzyć ten wzorzec. Gdy skończysz, spróbuj dodać trzeci argument do translate (), aby przenieść wzór na trzy wymiary.

```
size(200,200);

background(0);

stroke(255);

fill(255,100);

translate(_______,_______);

rect(0,0,100,100);

translate(_______,_______);

rect(0,0,100,100);

translate(_______,_______);

line(0,0,-50,50);
```

Funkcja translate() jest szczególnie przydatna, gdy rysujesz zbiór kształtów względem danego punktu środkowego. Wracając do Stworka z pierwszych 10 części, widzieliśmy taki kod:

```
void display() {

// Draw Zoog's body

fill(150);

rect( x,y ,w/6,h*2);

// Draw Zoog's head

fill(255);

ellipse( x,y-h/2 ,w,h);

}
```

Powyższa funkcja display() rysuje wszystkie części Zoog (ciało i głowę itp.) Względem położenia Zo, x, y. Wymaga użycia x i y zarówno w rect (), jak i ellipse (). translate () pozwala nam po prostu ustawić początek przetwarzania (0,0) w lokalizacji Zoog (x, y), a zatem narysować kształty względem (0,0).

```
void display() {

// Move origin (0,0) to (x,y)

translate(x,y);   //translate () może być użyty do narysowania zbioru kształtów względem danego
                  //punktu.

// Draw Zoog's body

fill(150);

rect( 0,0 ,w/6,h*2);

// Draw Zoog's head

fill(255);

ellipse(0,-h/2,w,h);

}
```

14.2 P3D vs. OPENGL

Jeśli przyjrzysz się uważnie przykładowi 14-3, zauważysz, że dodaliśmy trzeci argument do funkcji size (). Tradycyjnie size () służyło jednemu celowi: określić szerokość i wysokość naszego okna przetwarzania. Funkcja esize () akceptuje jednak również trzeci parametr wskazujący tryb rysowania. Tryb informuje Processing, co robić za scenami podczas renderowania okna wyświetlania. Domyślny tryb (jeśli nie jest określony) to „JAVA2D", który wykorzystuje istniejące biblioteki Java 2D do rysowania kształtów, ustawiania kolorów i tak dalej. Nie musimy się martwić o to, jak to działa. Twórcy Processing zadbali o szczegóły. Jeśli chcemy zastosować tłumaczenie 3D, tryb JAVA2D nie będzie już wystarczający. Uruchomienie przykładu w trybie domyślnym powoduje następujący błąd: „translate (x, y, z) może być użyte tylko z OPENGL lub P3D, zamiast tego użyj translate (x, y). " Zamiast przełączać się na translację (x, y), chcemy wybrać inny tryb. Są dwie opcje:

• P3D —P3D to renderer 3D opracowany przez twórców przetwarzania. Należy również zauważyć, że antyaliasing (włączany za pomocą funkcji smooth ()) nie jest dostępny w P3D.

• OPENGL —OPENGL to mechanizm renderujący 3D wykorzystujący akceleracją sprzętową. Jeśli masz zainstalowaną na komputerze kartę graficzną zgodną z OpenGL (co jest prawie każdym komputerem), możesz użyć tego trybu. Chociaż w czasie pisania tej książki nadal istnieje kilka, mniejszych zagięć, które można opracować w tym trybie (może się okazać, że rzeczy wyglądają nieco inaczej w P3D i OPENGL), może okazać się wyjątkowo przydatne pod względem szybkości. Jeśli planujesz wyświetlać dużą liczbę kształtów na ekranie w oknie o wysokiej rozdzielczości, tryb ten będzie prawdopodobnie miał najlepszą wydajność. Aby określić tryb, dodaj trzeci argument we wszystkich wersjach do funkcji size ().

```
size (200 200); // używając domyślnego trybu JAVA2D

size (200,200, P3D); // używając P3D

size(200,200, OPENGL); // używając OPENGL
```

Używając trybu OPENGL, musisz także zaimportować bibliotekę OPENGL. Można to zrobić, wybierając opcję menu SKETCH → IMPORT LIBRARY lub ręcznie dodając następujący wiersz kodu na początku szkicu

```
import processing.opengl. *;
```

Ćwiczenie 14-2: Uruchom dowolny szkic przetwarzania w P3D, a następnie przełącz na OPENGL. Zauważasz jakąkolwiek różnicę?

14.3 Kształty wierzchołków

Do tej pory nasza zdolność do rysowania na ekranie została ograniczona do małej listy prymitywnych dwuwymiarowych kształtów: prostokątów, elips, trójkątów, linii i punktów. Jednak w niektórych projektach pożądane jest tworzenie własnych niestandardowych kształtów. Można to zrobić za pomocą funkcji beginShape (), endShape () i vertex (). Rozważmy prostokąt. Prostokąt w Przetwarzaniu jest zdefiniowany jako punkt odniesienia, jak również szerokość i wysokość.

```
rect (50,50,100,100);
```

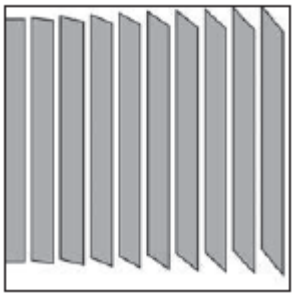
Ale możemy również uznać prostokąt za wielokąt (zamknięty kształt ograniczony segmentami linii) złożony z czterech punktów. Punkty wielokąta są nazywane wierzchołkami (liczba mnoga) lub wierzchołkiem (liczba pojedyncza). poniższy kod rysuje dokładnie ten sam kształt, co funkcja rect(), ustawiając wierzchołki prostokąta indywidualnie.

```
beginShape();

vertex(50,50);

vertex(150,50);

vertex(150,150);

vertex(50,150);

endShape(CLOSE);
```

beginShape () wskazuje, że utworzymy niestandardowy kształt składający się z pewnej liczby punktów wierzchołków: pojedynczy wielokąt. vertex () określa punkty dla każdego wierzchołka w wielokącie, a endShape () wskazuje, że jesteśmy gotowi do dodawania wierzchołków. Argument „CLOSE" wewnątrz endShape (CLOSE) deklaruje, że kształt powinien zostać zamknięty, to znaczy, że ostatni punkt wierzchołka powinien połączyć się z pierwszym. Dobrą rzeczą w używaniu niestandardowego kształtu nad prostym prostokątem jest elastyczność. Na przykład boki nie muszą być prostopadłe.

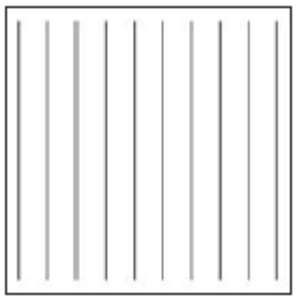
```
stroke(0);

fill(175);

beginShape();

vertex(50,50);

vertex(150,25);

vertex(150,175);

vertex(25,150);

endShape(CLOSE);
```

Mamy również możliwość tworzenia więcej niż jednego kształtu, na przykład w pętli, jak pokazano na rysunku



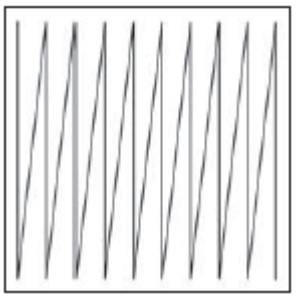
```
stroke(0);

for (int i = 0; i < 10; i + + ) {

beginShape();

fill(175);

vertex(i*20,10-i);

vertex(i*20 + 15,10 + i);

vertex(i*20 + 15,180 + i);

vertex(i*20,180-i);

endShape(CLOSE);

}
```

Możesz także dodać argument do beginShape () określający dokładnie, jaki kształt chcesz utworzyć. Jest to szczególnie przydatne, jeśli chcesz utworzyć więcej niż jeden wielokąt. Na przykład, jeśli utworzysz sześć punktów wierzchołków, nie ma możliwości, aby Przetwarzanie wiedziało, że naprawdę chcesz narysować dwa trójkąty (w przeciwieństwie do jednego sześciokąta), chyba że powiesz beginShape (TRIANGLES). Jeśli nie chcesz w ogóle tworzyć wielokąta, ale chcesz rysować punkty lub linie, możesz to powiedzieć, mówiąc beginShape (POINTS) lub beginShape (LINES).



```
stroke(0);

beginShape(LINES);

for (int i = 10; i < width; i + = 20) {

vertex(i,10);

vertex(i,height–10);

}

endShape();
```

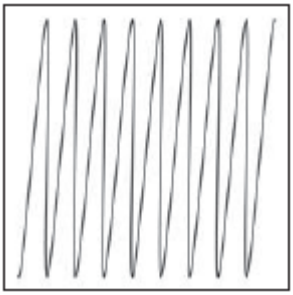
Zauważ, że LINIE są przeznaczone do rysowania szeregu pojedynczych linii, a nie ciągłej pętli. W przypadku ciągłej pętli nie używaj żadnego argumentu. Zamiast tego po prostu określ wszystkie potrzebne punkty wierzchołka i włącz noFill ().



```
noFill();

stroke(0);

beginShape();

for (int i = 10; i < width; i + = 20) {

vertex(i,10);

vertex(i,height–10);

}

endShape();
```

Pełna lista możliwych argumentów dla beginShape () jest dostępna w referencjach Processing : http://processing.org/reference/beginShape_.html

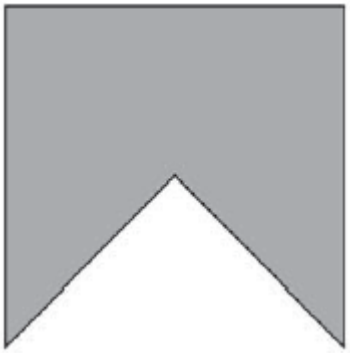
Ponadto wierzchołek () można zastąpić krzywą Vertex (), aby połączyć punkty z krzywymi zamiast linii prostych. Za pomocą curveVertex () zanotuj, jak nie są wyświetlane pierwsze i ostatnie punkty. Dzieje się tak dlatego, że muszą określić krzywiznę linii, gdy zaczyna się ona w drugim punkcie i kończy na drugim do ostatniego punktu



```
noFill();
stroke(0);
beginShape();
for (int i = 10; i < width; i + = 20) {
curveVertex(i,10);
curveVertex(i,height-10);
}
endShape();
```

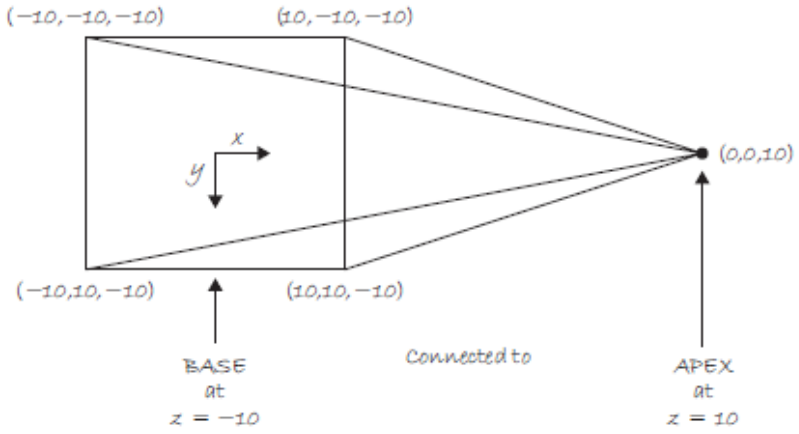
Ćwiczenie 14-3: Uzupełnij wierzchołki dla przedstawionego kształtu.

```
size(200,200);
background(255);
stroke(0);
fill(175);
beginShape();
vertex(20, 20);
vertex(_____,_____);
vertex(_____,_____);
vertex(_____,_____);
vertex(_____,_____);
endShape(_____);
```



14.4 Niestandardowe kształty 3D

Trójwymiarowe kształty można tworzyć za pomocą beginShape (), endShape () i vertex (), umieszczając wiele wielokątów obok siebie w odpowiedniej konfiguracji. Powiedzmy, że chcemy narysować czterostronną piramidę złożoną z czterech trójkątów, wszystkie połączone z jednym punktem (wierzchołkiem) i płaską płaszczyzną (podstawą). Jeśli Twój kształt jest wystarczająco prosty, możesz sobie poradzić z wypisaniem kodu. W większości przypadków, najlepiej jednak zacząć szkicować ołówkiem i papierem, aby określić położenie wszystkich wierzchołków. Jeden przykład naszej piramidy pokazano na rysunku



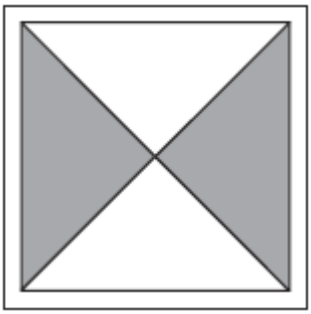

```
vertex(-10,-10,-10);   vertex(10,-10,-10);   vertex( 10,10,-10);   vertex(-10, 10,-10);
vertex( 10,-10,-10);   vertex(10, 10,-10);   vertex(-10,10,-10);   vertex(-10,-10,-10);
vertex(  0,   0, 10);   vertex( 0,   0, 10);   vertex(  0, 0, 10);   vertex(  0,   0, 10);
```

Przykład 14-4 pobiera wierzchołki z rysunku powyżej i umieszcza je w funkcji, która pozwala nam narysować piramidę o dowolnym rozmiarze. (Jako ćwiczenie spróbuj zrobić piramidę w obiekt.)

Przykład 14-4: Piramida za pomocą beginShape (TRIANGLES)

```
void setup() {

size(200,200,P3D);

}

void draw() {

background(255);
```

```
translate(100,100,0);
drawPyramid(150);      // Ponieważ wierzchołki piramidy są rysowane względem punktu środkowego,
                       //musimy wywołać translate (), aby poprawnie umieścić piramidę w oknie.
}
void drawPyramid(int t) {      // Funkcja ustawia wierzchołki piramidy wokół punktu środkowego na
                               //elastycznej odległości, w zależności od liczby przekazanej jako
                               //argument.
stroke(0);
// this pyramid has 4 sides, each drawn as a separate triangle
// each side has 3 vertices, making up a triangle shape
// the parameter " t " determines the size of the pyramid
beginShape(TRIANGLES);
fill(255,150);         // Zauważ, że każdy wielokąt może mieć własny kolor.
vertex(–t,–t,–t);
vertex( t,-t,–t);
vertex( 0, 0, t);
fill(150,150);
vertex( t,–t,–t);
vertex( t, t,–t);
vertex( 0, 0, t);
fill(255,150);
vertex( t, t,–t);
vertex(–t, t,–t);
vertex( 0, 0, t);
fill(150,150);
vertex(–t, t,–t);
vertex(–t,–t,–t);
vertex( 0, 0, t);
endShape();
}
```



Ćwiczenie 14-4: Utwórz piramidę tylko z trzema bokami. Dołącz bazę (w sumie cztery trójkąty). Użyj spacji poniżej, aby naszkicować lokalizacje wierzchołków jak na rysunku piramidy powyżej

Ćwiczenie 14-5: Utwórz trójwymiarową kostkę za pomocą ośmiu quadów— beginShape (QUADS). (Zauważ, że prostszym sposobem utworzenia kostki w Przetwarzaniu jest funkcja box ().)

14.5 Prosty obrót

Nie ma nic szczególnie trójwymiarowego w wizualnym wyniku przykładu piramidy. Obraz wygląda bardziej jak prostokąt z dwoma liniami połączonymi ukośnie od końca do końca. Znów mamy przypomnieć sobie, że tworzymy jedynie trójwymiarową iluzję i nie jest ona szczególnie skuteczna bez animowania struktury piramidy w przestrzeni wirtualnej. Jednym ze sposobów wykazania różnicy byłoby obrócenie piramidy. Dowiedzmy się więc o rotacji. Dla nas, w naszym świecie fizycznym, rotacja jest całkiem prostą i intuicyjną koncepcją. Chwyć pałeczkę, obróć ją i poczuj, co to znaczy obracać obiekt. Obracanie programowania niestety nie jest takie proste. Pojawiają się różnego rodzaju pytania. Wokół której osi należy obracać? Pod jakim kątem? Wokół jakiego punktu początkowego? Przetwarzanie oferuje kilka funkcji związanych z rotacją, które będziemy badać powoli, krok po kroku. Naszym celem będzie zaprogramowanie symulacji układu słonecznego z wieloma planetami obracającymi się wokół gwiazdy przy różnych prędkościach (jak również w celu obrócenia naszej piramidy w celu lepiej doświadcz jego trójwymiarowości). Ale najpierw spróbujmy czegoś prostego i spróbujmy obrócić jeden prostokąt wokół jego środka. Powinniśmy być w stanie uzyskać rotację z następującymi trzema zasadami:

1. Kształty są obracane podczas przetwarzania za pomocą funkcji rotate ().

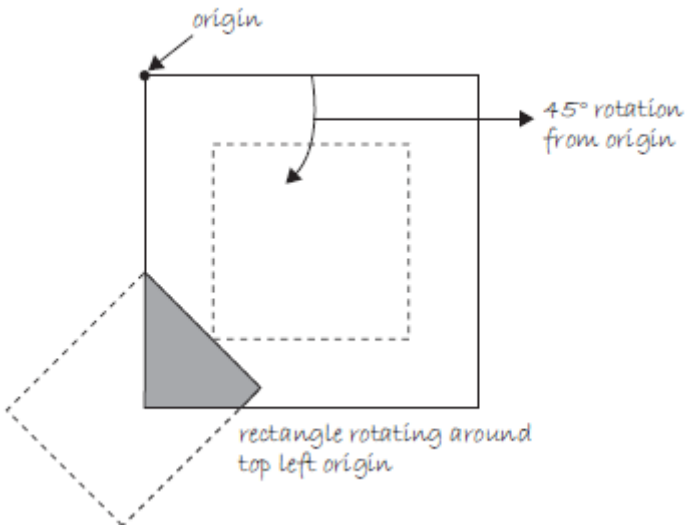
2. Funkcja rotate () przyjmuje jeden argument, kąt zmierzony w radianach.

3. obróć () obróci kształt w kierunku zgodnym z ruchem wskazówek zegara (w prawo).

OK, uzbrojeni w tę wiedzę powinniśmy być w stanie wywołać funkcję rotate () i przejść pod kątem. Powiedz: 45 ° (lub PI / 4 radiany) w przetwarzaniu. Oto nasza pierwsza (aczkolwiek wadliwa) próba, z wyjściem pokazanym na rysunku

```
rotate(radians(45));

rectMode(CENTER);

rect(width/2,height/2,100,100);
```

Strzelać. Co poszło nie tak? Prostokąt wygląda na obrócony, ale jest w złym miejscu! Najważniejszym faktem, o którym należy pamiętać przy obracaniu w przetwarzaniu, jest to, że kształty zawsze obracają się wokół punktu początkowego. Gdzie jest punkt początkowy w tym przykładzie? W lewym górnym rogu! Pochodzenie nie zostało przetłumaczone. Prostokąt nie będzie więc obracał się wokół własnego środka. Zamiast tego obraca się wokół lewego górnego rogu.



Jasne, może nadejść dzień, kiedy wszystko, co chcesz zrobić, to obrócić kształty wokół lewego górnego rogu, ale do tego dnia zawsze będziesz musiał najpierw przesunąć początek do odpowiedniego miejsca przed obróceniem, a następnie wyświetlić prostokąt.

```
translate(width/2,height/2);

rotate(radians(45));

rectMode(CENTER);

rect(0,0,100,100);      // Ponieważ przetłumaczyliśmy, aby obrócić, prostokąt żyje teraz w punkcie
                        //(0,0).
```

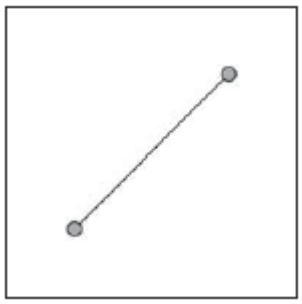
Możemy rozwinąć powyższy kod za pomocą lokalizacji mouseX, aby obliczyć kąt obrotu, a tym samym animować prostokąt, umożliwiając jego obrót. Patrz przykład 14-5.

Przykład 14-5: Prostokąt obracający się wokół środka

```
void setup() {

size(200,200);

}

void draw() {

background(255);
```

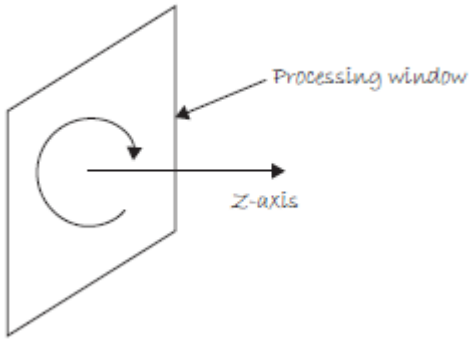
```
stroke(0);

fill(175);

// Translate origin to center

translate(width/2,height/2);

// theta is a common name of a variable to store an angle

float theta = PI*mouseX / width;        // Kąt waha się od 0 do PI w oparciu o stosunek lokalizacji
                                        //mouseX do szerokości szkicu

// Rotate by the angle theta

rotate(theta);

// Display rectangle with CENTER mode

rectMode(CENTER);

rect(0,0,100,100);

}
```

Ćwiczenie 14-6: Utwórz linię, która obraca się wokół jej środka (jak kręcenie pałeczką). Narysuj okrąg na obu punktach końcowych.



14.6 Obrót wokół różnych osi

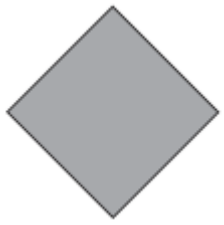
Teraz, gdy mamy już podstawową rotację, możemy zacząć zadawać następne ważne pytanie dotyczące rotacji: wokół której osi chcemy obracać? W poprzedniej części nasz kwadrat obracał się wokół osi Z. Jest to domyślna oś obrotu dwuwymiarowego.



Processing pozwoli również na obrót wokół osi x lub y za pomocą funkcji rotateX () i rotateY (), z których każda wymaga trybu P3D lub OPENGL. Funkcja rotateZ () również istnieje i jest odpowiednikiem rotate ().

Przykład 14-6: rotateZ

```
float theta = 0.0;
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,
height/2);
rotateZ(theta);
rectMode(CENTER);
rect(0,0,100,100);
theta + = 0.02;
}
```



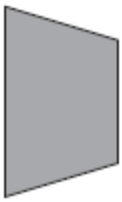
Przykład 14-7: rotateX

```
loat theta = 0.0;
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
```

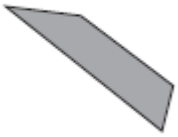
```
fill(175);
translate(width/2,
height/2);
rotateX(theta);
rectMode(CENTER);
rect(0,0,100,100);
theta + = 0.02;
}
```

Przykład 14-8: rotateY

```
float theta = 0.0;
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,
height/2);
rotateY(theta);
rectMode(CENTER);
rect(0,0,100,100);
theta + = 0.02;
}
```



Funkcje obracania można również stosować w połączeniu. Wyniki przykładu 14-9 pokazano na rysunku



Przykład 14-9: Obróć wokół więcej niż jednej osi

```
void setup() {
size(200,200,P3D);
}
void draw() {
background(255);
stroke(0);
fill(175);
translate(width/2,height/2);
rotateX(PI*mouseY/height);
rotateY(PI*mouseX/width);
rectMode(CENTER);
rect(0,0,100,100);
}
```

Wracając do przykładu piramidy, zobaczymy, jak obracanie sprawia, że trójwymiarowa jakość kształtu staje się bardziej widoczna. Przykład jest również rozszerzony o drugą piramidę, która jest wyłączona z pierwszej piramidy za pomocą translacji. Zauważ jednak, że obraca się wokół tego samego punktu początkowego co pierwsza piramida (ponieważ rotateX () i rotateY () są wywoływane przed drugim tłumaczeniem ()).

Przykład 14-10: Piramida

```
float theta = 0.0;
void setup() {
size(200,200,P3D);
```

```
}
void draw() {
background(144);
theta + = 0.01;
translate(100,100,0);
rotateX(theta);
rotateY(theta);
drawPyramid(50);
// translate the scene again
translate(50,50,20);
// call the pyramid drawing function
drawPyramid(10);
}
void drawPyramid(int t) {
stroke(0);
// this pyramid has 4 sides, each drawn as a separate triangle
// each side has 3 vertices, making up a triangle shape
// the parameter "t" determines the size of the pyramid
fill(150,0,0,127);
beginShape(TRIANGLES);
vertex(–t,–t,–t);
vertex( t,–t,–t);
vertex( 0, 0, t);
fill(0,150,0,127);
vertex( t,–t,–t);
vertex( t, t,–t);
vertex( 0, 0, t);
fill(0,0,150,127);
vertex( t, t,–t);
vertex(–t, t,–t);
vertex( 0, 0, t);
```

```
fill(150,0,150,127);

vertex(–t, t,–t);

vertex(–t,–t,–t);

vertex( 0, 0, t);

endShape();

}
```

Ćwiczenie 14-7: Obróć kostkę 3D wykonaną w ćwiczeniu 14-5. Czy możesz go obrócić w rogu lub w środku? Możesz także użyć pola funkcji Przetwarzanie (), aby utworzyć kostkę.

Ćwiczenie 14-8: Zrób klasę Pyramid

14.7 Skala

Oprócz translate () i rotate () istnieje jeszcze jedna funkcja, scale (), która wpływa na sposób orientacji i rysowania kształtów na ekranie. scale () zwiększa lub zmniejsza rozmiar obiektów na ekranie. Podobnie jak w przypadku obracania (), efekt skalowania jest wykonywany względem lokalizacji pochodzenia. scale () przyjmuje wartość punktu flotowania, procent, w którym skala: 1,0 wynosi 100%. Na przykład skala (0,5) rysuje obiekt w 50% jego wielkości i skali (3.0) zwiększa rozmiar obiektu do 300%. Poniżej znajduje się powtórzenie przykładu 14-1 (rosnący kwadrat) za pomocą skali ().

Przykład 14-11: Rosnący prostokąt przy użyciu scale()

```
float r = 0.0;

void setup() {

size(200,200);

}

void draw() {

background(0);

// Translate to center of window

translate(width/2,height/2);

// Scale any shapes according to value of r

scale(r);       // scale () zwiększa wymiary obiektu względem początku o procent (1,0 = 100%).
                //Zauważ, że w tym przykładzie efekt skalowania powoduje, że kontur kształtu staje
                //się grubszy.

// Display a rectangle in the middle of the screen

stroke(255);

fill(100);

rectMode(CENTER);

rect(0,0,10,10);
```

// Increase the scale variable

r + = 0.02;

}

scale () może również przyjmować dwa argumenty (do skalowania wzdłuż osi x i y z różnymi wartościami różnicy) lub trzy argumenty (dla osi x -, y - i z).

14.8 Matryca: pchanie i strzelanie

Czym jest macierz?

Aby śledzić rotacje i tłumaczenia oraz jak wyświetlać kształty według różnych transformacji, Przetwarzanie (i prawie każde oprogramowanie grafiki komputerowej) używa matrycy. Jak działa transformacja macierzy wykracza poza zakres tego tekstu; Jednak dobrze jest po prostu wiedzieć, że informacje dotyczące układu współrzędnych są przechowywane w tak zwanej macierzy transformacji. Po zastosowaniu translacji lub obrotu zmienia się macierz transformacji. Od czasu do czasu warto zapisać bieżący stan macierzy, która ma zostać przywrócona później. To ostatecznie pozwoli nam na poruszanie i obracanie poszczególnych kształtów bez wpływania na innych.

Czym jest macierz?

Macierz to tabela liczb z wierszami i kolumnami. W przetwarzaniu macierz transformacji jest używana do opisu orientacji okna - czy jest ona tłumaczona czy obracana? Możesz wyświetlić bieżącą macierz w dowolnym momencie, wywołując funkcję printMatrix (). Tak wygląda macierz w jej „normalnym" stanie, bez wywołań translate () lub rotate ().

1,0000 0,0000 0,0000

0,0000 1,0000 0,0000

Ta koncepcja najlepiej ilustruje przykład. Dajmy sobie przypisanie: Utwórz szkic przetwarzania, w którym dwa prostokąty obracają się z różnymi prędkościami w różnych kierunkach wokół ich odpowiednich punktów środkowych. Gdy zaczniemy rozwijać ten przykład, zobaczymy, gdzie pojawiają się problemy i jak będziemy musieli zaimplementować funkcje pushMatrix () i popMatrix (). Począwszy od zasadniczo tego samego kodu z sekcji 14.4, możemy obracać kwadrat wokół osi Z w lewym górnym rogu okna.

Przykład 14-12: Obracanie jednego kwadratu

```
float theta1 = 0;

void setup() {

size(200,200,P3D);

}

void draw() {

background (255);

stroke(0);

fill(175);
```

```
rectMode(CENTER);
translate(50,50);
rotateZ(theta1);
rect(0,0,60,60);
theta1 + = 0.02;
}
```

Dokonując drobnych korekt, teraz wprowadzamy obrotowy kwadrat w dolnym prawym rogu.

Przykład 14-13: Obracanie innego kwadratu

```
float theta2 = 0;
void setup() {
size(200,200,P3D);
}
void draw() {
background (255);
stroke(0);
fill(175);
rectMode(CENTER);
translate(150,150);
rotateY(theta2);
rect(0,0,60,60);
theta2 + = 0.02;
}
```

Bez starannego rozważenia możemy pomyśleć, aby po prostu połączyć dwa programy. Funkcja setup () pozostałaby taka sama, włączylibyśmy dwa globale, theta1 i theta2 i wywołali odpowiednie tłumaczenie i obrót dla każdego prostokąta. Dostosowalibyśmy również tłumaczenie drugiego kwadratu z tłumaczenia (150,150) na tłumaczenie (100, 100), ponieważ już przetłumaczyliśmy na (50,50) z pierwszym kwadratem.

```
float theta1 = 0;
float theta2 = 0;
void setup() {
size(200,200,P3D);
}
```

```
void draw() {

background(255);

stroke(0);

fill(175);

rectMode(CENTER);

translate(50,50);       // Pierwsze wywołanie rotateZ () wpływa na wszystkie później narysowane
                        //kształty. Oba kwadraty obracają się wokół środka pierwszego kwadratu.

rotateZ(theta1);

rect(0,0,60,60);

theta1 + = 0.02;

translate(100,100);

rotateY(theta2);

rect(0,0,60,60);

theta2 + = 0.02;

}
```

Uruchomienie tego przykładu szybko ujawni problem. Pierwszy (lewy górny) kwadrat obraca się wokół jego środka. Jednakże, podczas gdy drugi kwadrat obraca się wokół swojego środka, obraca się również wokół pierwszego kwadratu!

Pamiętaj, że wszystkie wywołania do tłumaczenia i obracania odnoszą się do poprzedniego stanu układu współrzędnych. Potrzebujemy sposobu na przywrócenie macierzy do jej pierwotnego stanu, tak aby poszczególne kształty mogły działać niezależnie. Zapisywanie i przywracanie stanu rotacji / translacji odbywa się za pomocą funkcji pushMatrix () i popMatrix (). Aby zacząć, pomyślmy o nich jako saveMatrix () i restoreMatrix (). (Zauważ, że nie ma takich funkcji.) Push = zapisz. Pop = przywróć.

Aby każdy kwadrat obracał się sam, możemy napisać następujący algorytm (z nowymi częściami pogrubionymi).

1. Zapisz bieżącą macierz transformacji. To jest miejsce, w którym zaczęliśmy, z (0,0) w lewym górnym rogu okna i bez obrotu.

2. Przetłumacz i obróć pierwszy prostokąt.

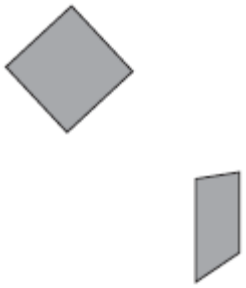
3. Wyświetl pierwszy prostokąt.

4. Przywróć macierz z kroku 1, aby nie dotyczyły jej kroki 2 i 3!

5. Przetłumacz i obróć drugi prostokąt.

6. Wyświetl drugi prostokąt.

Przepisanie naszego kodu w przykładzie 14-14 daje poprawny wynik, jak pokazano na rysunku



Przykład 14-14: Obracanie obu kwadratów

```
float theta1 = 0;
float theta2 = 0;

void setup() {
  size(200,200,P3D);
}

void draw() {
  background(255);
  stroke(0);
  fill(175);
  rectMode(CENTER);

  pushMatrix();                    1

  translate(50,50);                2
  rotateZ(theta1);
  rect(0,0,60,60);                 3

  popMatrix();                     4

  pushMatrix();
  translate(150,150);              5
  rotateY(theta2);
  rect(0,0,60,60);                 6
  popMatrix();

  theta1 += 0.02;
  theta2 += 0.02;
}
```

Chociaż technicznie nie jest to wymagane, dobrym zwyczajem jest umieszczanie pushMatrix () i popMatrix () wokół drugiego prostokąta (na wypadek gdybyśmy mieli dodać więcej do tego kodu). Dobra zasada, kiedy zaczyna się od użycia pushMatrix () i popMatrix () przed i po tłumaczeniu i obrocie dla wszystkich kształtów, aby można je było traktować jako pojedyncze elementy. W rzeczywistości ten przykład powinien być zorientowany obiektowo, z każdym obiektem wykonującym własne wywołania pushMatrix (), translate (), rotate () i popMatrix ().

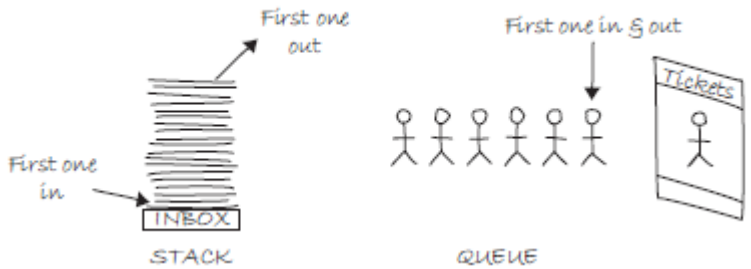
Przykład 14-15: Obracanie wielu rzeczy za pomocą obiektów

```
// An array of Rotater objects

Rotater[] rotaters;

void setup() {
```

```
size(200,200);
rotaters = new Rotater[20];
// Rotaters are made randomly
for (int i = 0; i < rotaters.length; i + + ) {
rotaters[i] = new Rotater(random(width),random(height),random(–0.1,0.1),random(48));
}
}
void draw() {
background(255);
// All Rotaters spin and are displayed
for (int i = 0; i < rotaters.length; i + + ) {
rotaters[i].spin();
rotaters[i].display();
}
}
// A Rotater class
class Rotater {
float x,y; // x,y location
float theta; // angle of rotation
float speed; // speed of rotation
float w; // size of rectangle
Rotater(float tempX, float tempY, float tempSpeed, float tempW) {
x = tempX;
y = tempY;
theta = 0; // Angle is always initialized to 0
speed = tempSpeed;
w = tempW;
}
// Increment angle
void spin() {
theta + = speed;
```

```
}

// Display rectangle

void display() {

rectMode(CENTER);

stroke(0);

fill(0,100);

// Note the use of pushMatrix()

// and popMatrix() inside the object's

// display method!

pushMatrix();   // pushMatrix () i popMatrix () są wywoływane wewnątrz metody display () klasy. W
                //ten sposób każdy obiekt Rotater jest renderowany z własnym niezależnym
                //tłumaczeniem i obrotem!

translate(x,y);

rotate(theta);

rect(0,0,w,w);

popMatrix();

}

}
```

Interesujące wyniki można również uzyskać z zagnieżdżania wielu wywołań do pushMatrix () i popMatrix (). Zawsze musi istnieć taka sama liczba wywołań zarówno do pushMatrix (), jak i popMatrix (), ale nie zawsze muszą one przychodzić jedna po drugiej. Aby zrozumieć, jak to działa, przyjrzyjmy się bliżej znaczeniu „wypychania" i „popu. „Push" i „pop" odnoszą się do koncepcji w informatyce, znanej jako stos. Wiedza o tym, jak działa stos, pomoże Ci prawidłowo używać pushMatrix () i popMatrix (). Stos jest dokładnie taki: stos. Rozważmy, jak nauczyciel angielskiego osiedlił się na noc oceniania dokumentów przechowywanych w stosie na biurku, na stosie papierów. Nauczyciel układa je jeden po drugim i odczytuje je w odwrotnej kolejności niż stos. Pierwszy papier umieszczony na stosie jest ostatnim czytanym. Ostatni dodany papier jest pierwszym czytanym. Zauważ, że jest to dokładne przeciwieństwo kolejki. Jeśli czekasz w kolejce na zakup biletów na film, pierwszą osobą w kolejce jest pierwsza osoba, która ma kupić bilety, ostatnia osoba jest ostatnia.



Pchanie odnosi się do procesu umieszczania czegoś w stosie, wyskakiwania z czegoś. Dlatego zawsze musisz mieć równą liczbę wywołań pushMatrix () i popMatrix (). Nie możesz popchnąć czegoś, jeśli nie istnieje! (Jeśli masz niepoprawną liczbę popchnięć i wyskoków, Przetwarzanie powie: „Zbyt wiele wywołań do popMatrix () (i za mało do pushMatrix)". Korzystając z programu obracających się kwadratów jako podstawy, możemy zobaczyć, jak zagnieżdżanie pushMatrix () i popMatrix () jest użyteczny.Następujący szkic ma jedno koło w środku (nazwijmy je słońcem) z innym okręgiem obracającym go (nazwijmy to ziemią) i kolejne dwa obracające się wokół niego (nazwijmy je księżycem # 1 i księżycem # 2).

Przykład 14-16: Prosty układ słoneczny

```
// Angle of rotation around sun and planets
float theta = 0;
void setup() {
size(200,200);
smooth();
}
void draw() {
background(255);
stroke(0);
// Translate to center of window
// to draw the sun.
translate(width/2,height/2);
fill(255,200,50);
ellipse(0,0,20,20);
// The earth rotates around the sun
pushMatrix();
rotate(theta);
translate(50,0);
fill(50,200,255);
ellipse(0,0,10,10);
// Moon #1 rotates around the earth
pushMatrix();
rotate(–theta*4);     // pushMatrix () jest wywoływany, aby zapisać stan transformacji przed
                      //narysowaniem księżyca # 1. W ten sposób możemy pop i powrócić na ziemię
                      //przed narysowaniem księżyca # 2. Oba księżyce obracają się wokół Ziemi
                      //(która sama się obraca wokół Słońca).
```

```
translate(15,0);

fill(50,255,200);

ellipse(0,0,6,6);

popMatrix();

// Moon #2 also rotates around the earth

pushMatrix();

rotate(theta*2);

translate(25,0);

fill(50,255,200);

ellipse(0,0,6,6);

popMatrix();

popMatrix();

theta + = 0.01;

}
```

pushMatrix () i popMatrix () mogą być również zagnieżdżone wewnątrz pętli for lub while z dość unikalnymi i interesującymi wynikami. Poniższy przykład jest trochę łamigłówką, ale zachęcam do zabawy.

Przykład 14-17: Zagnieżdżone push i pop

```
// Global angle for rotation

float theta = 0;

void setup() {

size(200, 200);

smooth();

}

void draw() {

background(100);

stroke(255);

// Translate to center of window

translate(width/2,height/2);

// Loop from 0 to 360 degrees (2*PI radians)

for(float i = 0; i < TWO_PI; i + = 0.2) {

// Push, rotate and draw a line!
```

```
pushMatrix();

rotate(theta + i);

line(0,0,100,0);

// Loop from 0 to 360 degrees (2*PI radians)

for(float j = 0; j < TWO_PI; j + = 0.5) {

// Push, translate, rotate and draw a line!

pushMatrix();

translate(100,0);      // Stan transformacji jest zapisywany na początku każdego cyklu przez pętlę
                       //for i przywrócony na końcu. Spróbuj skomentować te linie, aby zobaczyć
                       //różnicę!

rotate(–theta–j);

line(0,0,50,0);

// We're done with the inside loop, pop!

popMatrix();

}

// We're done with the outside loop, pop!

popMatrix();

}

endShape();

// Increment theta

theta + = 0.01;

}
```

Ćwiczenie 14-9: Weź piramidę lub kształt kostki i uczyń ją klasą. Niech każdy obiekt wywoła własne wywołanie pushMatrix () i popMatrix (). Czy możesz stworzyć tablicę obiektów obracających się niezależnie w 3D?

14.9 Układ Słoneczny w Processing

Korzystając z wszystkich technik translacji, rotacji, pchania i wyskakiwania w tym rozdziale, jesteśmy gotowi zbudować układ słoneczny przetwarzający. Przykładem będzie zaktualizowana wersja przykładu 14-16 z poprzedniej sekcji (bez księżyców), z dwiema głównymi zmianami:

• Każda planeta będzie obiektem, członkiem klasy Planet.

• Szereg planet okrąży Słońce.

Przykład 14-18: Obiektowy układ słoneczny

```
// An array of 8 planet objects
```

```
Planet[] planets = new Planet[8];
void setup() {
size(200,200);
smooth();
// The planet objects are initialized using the counter variable
for (int i = 0; i < planets.length; i + + ) {
planets[i] = new Planet(20 + i*10,i + 8);
}
}
void draw() {
background(255);
// Drawing the Sun
pushMatrix();
translate(width/2,height/2);
stroke(0);
fill(255);
ellipse(0,0,20,20);
// Drawing all Planets
for (int i = 0; i < planets.length; i + + ) {
planets[i].update();
planets[i].display();
}
popMatrix();
}
class Planet {
float theta; // Rotation around sun        // Każdy obiekt planety śledzi swój własny kąt obrotu.
float diameter; // Size of planet
float distance; // Distance from sun
float orbitspeed; // Orbit speed
Planet(float distance_, float diameter_) {
distance = distance_;
```

```
diameter = diameter_;

theta = 0;

orbitspeed = random(0.01,0.03);

}

void update() {

// Increment the angle to rotate

theta + = orbitspeed;

}

void display() {              //Przed obrotem i tłumaczeniem zapisywany jest stan macierzy z pushMatrix().

pushMatrix();

rotate(theta); // rotate orbit

translate(distance,0); // translate out distance

stroke(0);

fill(175);

ellipse(0,0,diameter,diameter);

// Afer we are done, restore matrix!

popMatrix();     // Po narysowaniu planety macierz zostaje przywrócona za pomocą popMatrix (), aby
                 //nie wpłynąć na następną planetę.

}

}
```

Ćwiczenie 14-10: Jak dodasz księżyce do planet? Podpowiedź: Napisz klasę Księżyca, która jest praktycznie identyczna z planetą. Następnie włącz zmienną Księżyca do klasy Planet.

Ćwiczenie 14-11: Rozszerz przykład układu słonecznego na trzy wymiary. Spróbuj użyć sphere () lub box () zamiast elipsy (). Uwaga: sphere () przyjmuje jeden argument, promień kuli. box () może przyjąć jeden argument (rozmiar, w przypadku kostki) lub trzy argumenty (szerokość, wysokość i głębokość).

Projekt lekcji szóstej

Stwórz wirtualny ekosystem. Stwórz klasę dla każdego „stworzenia" w twoim świecie. Używając technik z części 13 i 14, spróbuj tchnąć swoje stworzenia w osobowość. Niektóre możliwości:

• Użyj szumu Perlina do kontrolowania ruchów istot.

• Spraw, by stworzenia wyglądały, jakby oddychały z oscylacją.

• Zaprojektuj stwory za pomocą rekursji.

• Zaprojektuj niestandardowe wielokąty za pomocą beginShape ().

• Używaj rotacji w zachowaniach stworzeń.

Użyj miejsca podanego poniżej do szkicowania projektów, notatek i pseudokodów dla swojego projektu.

VII Obrazy

Obraz cyfrowy to nic innego jak dane - liczby wskazujące na zmiany koloru czerwonego, zielonego i niebieskiego w określonym miejscu na siatce pikseli. Przez większość czasu oglądamy te piksele jako miniaturowe prostokąty umieszczone razem na ekranie komputera. Przy odrobinie kreatywnego myślenia i trochę manipulacji pikselami z kodem możemy jednak wyświetlać te informacje na wiele sposobów. Ta część poświęcona jest wyrwaniu prostego rysunku kształtu w Przetwarzaniu i użyciu obrazów (i ich pikseli) jako elementów konstrukcyjnych przetwarzania grafiki.

Rozpoczęcie pracy z obrazami

Do tej pory jesteśmy całkiem zadowoleni z idei typów danych. Często je określamy - zmienna zmienna o nazwie speed, int o nazwie x, a może nawet char o nazwie letterGrade. Są to wszystkie prymitywne typy danych, bity znajdujące się w pamięci komputera gotowe do użycia. Choć może jest to nieco trudniejsze, zaczynamy także czuć się swobodnie z obiektami, złożone typy danych, które przechowują wiele fragmentów danych (wraz z funkcjonalnością) - na przykład klasa Zoog zawierała zmienne punktowe dla lokalizacji, rozmiaru, i szybkość, a także metody poruszania się, wyświetlania się i tak dalej. Zoog jest oczywiście klasą zdefiniowaną przez użytkownika; wprowadziliśmy Zoog do tego świata programowania, definiując, co to znaczy być Zoogiem, i definiując dane i funkcje związane z obiektem Zoog. Oprócz obiektów zdefiniowanych przez użytkownika, przetwarzanie ma wiele przydatnych klas, które są gotowe do pracy bez pisania przez nas żadnego kodu. Pierwszą klasą zdefiniowaną przez Przetwarzanie, którą zbadamy, jest PImage, klasa do ładowania i wyświetlania obrazu, takiego jak ten pokazany na rysunku

```
// Deklarowanie zmiennej typu PImage

PImage img;     //Deklarowanie zmiennej typu PImage, klasy dostępnej dla nas z biblioteki rdzeniowej
                // Processing

void setup() {

size(320,240);

// Utwórz nową instancję PImage, ładując plik obrazu

img = loadImage( " mysummervacation.jpg " );

}

void draw() {

background(0);
```

```
image(img,0,0);          //Funkcja image () wyświetla obraz w lokalizacji - w tym przypadku w punkcie
                         //(0,0).
}
```

Użycie instancji obiektu PImage nie różni się od używania klasy zdefiniowanej przez użytkownika. Najpierw deklarowana jest zmienna typu PImage o nazwie „img". Po drugie, nowa instancja obiektu PImage jest tworzona za pomocą metody loadImage (). loadImage () pobiera jeden argument, łańcuch wskazujący nazwę pliku i ładuje ten plik do pamięci. loadImage () szuka plików graficznych przechowywanych w folderze danych szkicu przetwarzania.

Folder danych: Jak się tam dostać?

Obrazy można dodawać do folderu danych automatycznie przez:

Sketch → Add File…

lub ręcznie

Sketch → Show Sketch Folder

Otworzy się folder szkicu, jak pokazano na rysunku.

Jeśli nie ma katalogu danych, utwórz go. W przeciwnym razie umieść pliki obrazów w środku. Przetwarzanie akceptuje następujące formaty plików dla obrazów: GIF, JPG, TGA i PNG.

W przykładzie 15-1 może wydawać się nieco dziwne, że nigdy nie nazwaliśmy „konstruktora", aby utworzyć instancję obiektu PImage, mówiąc „new PImage ()". W końcu we wszystkich dotychczasowych przykładach związanych z obiektami konstruktor jest konieczny do utworzenia instancji obiektu.

Spaceship ss = new Spaceship();

Flower flr = new Flower(25);

I jeszcze:

PImage img = loadImage ("file.jpg");

W rzeczywistości funkcja loadImage () wykonuje pracę konstruktora, zwracając zupełnie nową instancję obiektu PImage wygenerowanego z podanej nazwy. Możemy myśleć o tym jako o

konstruktorze PImage dla ładowanie obrazów z pliku. Do utworzenia pustego obrazu używana jest funkcja createImage ().

// Utwórz pusty obraz, 200 x 200 pikseli z kolorem RGB

PImage img = createImage (200,200, RGB);

Powinniśmy również zauważyć, że proces ładowania obrazu z dysku twardego do pamięci jest powolny i powinniśmy upewnić się, że nasz program musi to zrobić tylko raz, w setup (). Ładowanie obrazów w draw () może skutkować niską wydajnością, a także błędami „Out of Memory". Po załadowaniu obrazu jest on wyświetlany za pomocą funkcji image (). Funkcja image () musi zawierać trzy argumenty - obraz, który ma zostać wyświetlony, lokalizację x i lokalizację y. Opcjonalnie dwa argumenty można dodać, aby zmienić rozmiar obrazu do określonej szerokości i wysokości.

image (img, 10,20,90,60);

Ćwiczenie 15-1: Załaduj i wyświetl obraz. Kontroluj szerokość i wysokość obrazu za pomocą myszy.

Animacja z obrazem

Stąd łatwo jest zobaczyć, w jaki sposób można wykorzystać obrazy do dalszego rozwijania przykładów z poprzednich części

Przykład 15-2: Obraz „sprite"

```
PImage head; // A variable for the image file

float x,y; // Variables for image location

float rot; // A variable for image rotation

void setup() {

size(200,200);

// load image, initialize variables

head = loadImage( "face.jpg");

x = 0.0f;

y = width/2.0f;

rot = 0.0f;

}

void draw() {

background(255);

// Translate and rotate

translate(x,y);

rotate(rot);

image(head,0,0);          // Draw image //Obrazy mogą być animowane tak jak zwykłe kształty za
                          //pomocą zmiennych, translate (), obracanie () i tak dalej.
```

// Dostosuj zmienne, aby utworzyć animację

x + = 1.0;

rot + = 0.01;

if (x > width) {

x = 0;

}

}

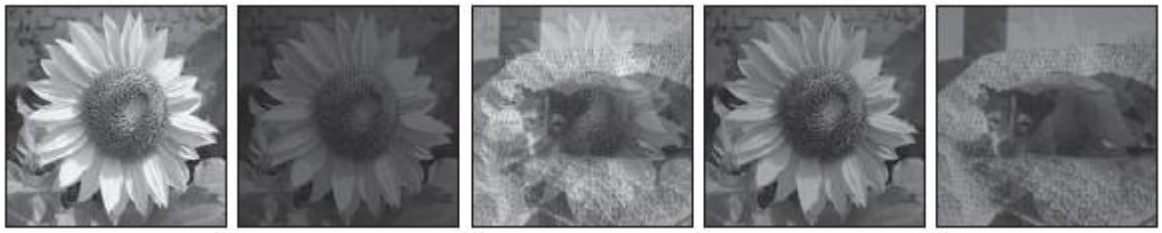
Ćwiczenie 15-2: Przepisz ten przykład w sposób zorientowany obiektowo, w którym dane obrazu, lokalizacji, rozmiaru, obrotu itd. Są zawarte w klasie. Czy możesz zamienić obrazy klas, gdy trafią w krawędź ekranu?

```
class Head {
  ________ // Avariable for the image file
  ________ // Variables for image location
  ________ // A variable for image rotation

  Head(String filename, ________, ________) {
    // Load image, initialize variables
    ________ = loadImage(________);
    ____________________________________
    ____________________________________
    ____________________________________
  }

  void display() {
    ____________________________________
    ____________________________________
    ____________________________________
  }

  void move() {
    ____________________________________
    ____________________________________
    ____________________________________
    ____________________________________
    ____________________________________
    ____________________________________
  }
}
```

Mój pierwszy filtr przetwarzania obrazu

Co jakiś czas, podczas wyświetlania obrazu, wybieramy zmianę jego wyglądu. Być może chcielibyśmy, aby obraz był ciemniejszy, przezroczysty, niebieskawy i tak dalej. Ten typ prostego filtrowania obrazu

jest uzyskiwany dzięki funkcji tint () Przetwarzania. tint () jest zasadniczo odpowiednikiem obrazu fill (), ustawiającym kolor i przezroczystość alfa do wyświetlania obrazu na ekranie. Niemniej jednak obraz zazwyczaj nie jest jednokolorowy. Argumenty dla tint () po prostu określają, ile danego koloru ma użyć dla każdego piksela tego koloru obraz, jak również jak przezroczyste powinny być te piksele. W poniższych przykładach założymy, że załadowano dwa obrazy (sunflower i pies), a pies jest wyświetlany jako tło (co pozwoli nam wykazać przezroczystość). Patrz rysunek.



Kolorowe wersje tych obrazów można znaleźć na stronie: http://www.learningprocessing.com)

```
PImage sunflower = loadImage( "sunflower.jpg");

PImage dog = loadImage( "dog.jpg");

background(dog);
```

Jeśli tint () otrzymuje jeden argument, dotyczy to tylko jasności obrazu.

```
tint(255);                  //A Obraz zachowuje swój pierwotny stan.

image(sunflower,0,0);

tint(100);

image(sunflower,0,0);   //B Obraz wydaje się ciemniejszy
```

Drugi argument zmieni przezroczystość alfa obrazu.

```
tint(255,127);              //C Obraz ma 50% nieprzeźroczystości

image(sunflower,0,0);
```

Trzy argumenty wpływają na jasność czerwonego, zielonego i niebieskiego komponentu każdego koloru.

```
tint(0,200,255)

image(sunflower,0,0);   //D Żadna z nich nie jest czerwona, większość z nich jest zielona, a wszystko
                        //jest niebieskie.
```

Wreszcie dodanie czwartego argumentu do metody manipuluje alfą (tak samo jak w przypadku dwóch argumentów). Nawiasem mówiąc, zakres wartości dla tint () można określić za pomocą colorMode ()

```
tint(255,0,0,100);

image(sunflower,0,0);   //E Obraz jest zabarwiony na czerwono i przezroczysty
```

Ćwiczenie 15-3: Wyświetl obraz używając tint (). Użyj lokalizacji myszy, aby kontrolować ilość odcienia czerwonego, zielonego i niebieskiego. Spróbuj również użyć odległości myszy od rogów lub środka.

Ćwiczenie 15-4: Używając tint (), utwórz montaż mieszanych obrazów. Co się dzieje, gdy nakładasz na siebie dużą liczbę obrazów, z których każda ma inną przezroczystość alfa? Czy możesz sprawić, by był interaktywny, aby inne obrazy zanikały i znikały?

Tablica obrazów

Jedno zdjęcie jest ładne, dobre miejsce do rozpoczęcia. Nie potrwa to długo, dopóki nie przejmie pokusa używania wielu obrazów. Tak, możemy śledzić wiele obrazów z wieloma zmiennymi, ale tutaj jest okazja do ponownego odkrycia mocy tablicy. Załóżmy, że mamy pięć obrazów i chcemy wyświetlić nowy obraz tła za każdym razem, gdy użytkownik kliknie myszą. Po pierwsze, ustawiamy tablicę obrazów jako zmienną globalną.

// Image Array

PImage[] images = new PImage[5];

Po drugie, ładujemy każdy plik obrazu do odpowiedniej lokalizacji w tablicy. Dzieje się tak w setup ().

// Loading Images into an Array

images[0] = loadImage(" cat.jpg ");

images[1] = loadImage(" mouse.jpg ");

images[2] = loadImage(" dog.jpg ");

images[3] = loadImage(" kangaroo.jpg ");

images[4] = loadImage(" porcupine.jpg ");

Oczywiście jest to trochę niezręczne. Ładowanie każdego obrazu indywidualnie nie jest strasznie eleganckie. Z pięcioma obrazami na pewno jest to łatwe do zarządzania, ale wyobraź sobie, że piszesz powyższy kod ze 100 obrazami. Jednym z rozwiązań jest przechowywanie nazw plików w tablicy String i użycie instrukcji for do inicjalizacji wszystkich elementów tablicy.

// Loading Images into an Array from an array of filenames

String[] filenames = { "cat.jpg " , " mouse.jpg" , " dog.jpg " , " kangaroo.jpg " , " porcupine.jpg ");

for (int i = 0; i < filenames.length; i + +) {

images[i] = loadImage(filenames[i]);

}

<u>Konkatenacja: nowy rodzaj dodawania</u>

Zazwyczaj znak plus (+) oznacza, dodaj. 2 + 2 = 4, prawda?

Z tekstem (zapisanym w łańcuchu, ujętym w cudzysłów), + oznacza konkatenację, czyli łączenie dwóch łańcuchów razem.

„Happy” + „Trails” = „Happy Trails”

„2” + „2” = „22”

Co więcej, jeśli po prostu spędzimy trochę czasu z naszych gorączkowych planów, aby zaplanować z wyprzedzeniem, numerując pliki obrazów („animal1.jpg”, „animal2.jpg”, „animal3.jpg” itd.), Możemy naprawdę uprościć kod:

```
// Loading images with numbered files

for (int i = 0; i < images.length; i + + ) {

images[i] = loadImage( "animal" + i + ".jpg");

}
```

Po załadowaniu obrazów należy narysować (). Wybieramy wyświetlanie jednego konkretnego obrazu, wybierając z tablicy odwołanie do indeksu („0” poniżej).

```
image(images[0],0,0);
```

Oczywiście kodowanie wartości indeksu jest głupie. Potrzebujemy zmiennej, aby dynamicznie wyświetlać inny obraz w danym momencie.

```
image (images [imageindex], 0,0);
```

Zmienna „imageindex” powinna być zadeklarowana jako zmienna globalna (typu integer). Jego wartość można zmienić w trakcie trwania programu. Pełna wersja jest pokazana w przykładzie 15-3.

Przykład 15-3: Zamiana obrazów

```
int maxImages = 10; // Total # of images

int imageIndex = 0; // Initial image to be displayed is the first

PImage[] images = new PImage[maxImages]; // The image array  //Deklarowanie tablicy obrazów.

void setup() {

size(200,200);

// Loading the images into the array

// Don't forget to put the JPG files in the data folder!

for (int i = 0; i < images.length; i + + ) {

images[i] = loadImage( "animal" + i + ".jpg"); //Ładowanie tablicy obrazów

}

}

void draw() {

image(images[imageIndex],0,0); // Displaying one image //Wyświetlanie jednego obrazu z tablicy.

}

void mousePressed() {

// A new image is picked randomly when the mouse is clicked

// Note the index to the array must be an integer!
```

```
imageIndex = int(random(images.length)); //Wybór nowego obrazu do wyświetlenia poprzez zmianę
                                          //zmiennej indeksu!
}
```

Aby odtworzyć obrazy w sekwencji jako animację, postępuj zgodnie z Przykładem 15-4.

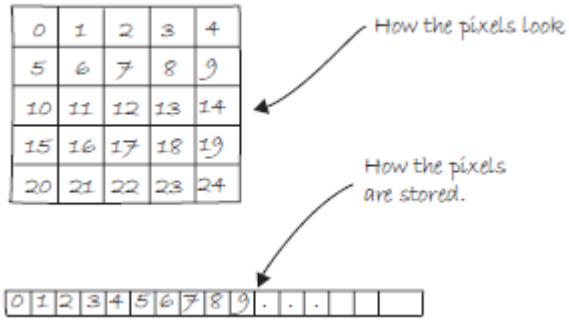
Przykład 15-4: Sekwencja obrazów

```
void draw() {
background(0);
image(images[imageIndex],0,0);
// increment image index by one each cycle
// use modulo "%" to return to 0 once the size
//of the array is reached
imageIndex = (imageIndex + 1) % images.length;    //Pamiętasz moduł? Znak % pozwala nam
                                                  //cyklicznie cofać licznik do 0.
}
```

Ćwiczenie 15-5: Utwórz wiele wystąpień sekwencji obrazów na ekranie. Niech zaczną się od innych czasów w sekwencji, aby nie były zsynchronizowane. Wskazówka: Użyj programowania obiektowego, aby umieścić sekwencję obrazów w klasie.

Pikseli, pikseli i więcej pikseli

Jeśli pilnie czytałeś tę książkę w dokładnie określonej kolejności, zauważysz, że jak dotąd jedynym dostępnym sposobem rysowania na ekranie jest wywołanie funkcji. „Narysuj linię między tymi punktami" lub „Wypełnij elipsę czerwonym" lub „załaduj ten obraz JPG i umieść go tutaj na ekranie. Ale gdzieś ktoś musiał napisać kod, który tłumaczy te wywołania funkcji na ustawienie poszczególnych pikseli na ekranie, aby odzwierciedlić żądany kształt. Linia nie pojawia się, ponieważ mówimy linia (), pojawia się, ponieważ kolor wszystkich pikseli wzdłuż liniowej ścieżki między dwoma punktami. Na szczęście nie musimy zarządzać ustawieniami piksela na niższym poziomie z dnia na dzień. Mamy twórców przetwarzania (i Java), aby podziękować za wiele funkcji rysowania, które zajmują się tym biznesem. Niemniej jednak od czasu do czasu chcemy wyrwać się z naszego przyziemnego rysowania kształtu i radzić sobie z pikselami na ekranie bezpośrednio. Przetwarzanie zapewnia tę funkcjonalność za pośrednictwem macierzy pikseli. Znamy pomysł, że każdy piksel na ekranie ma pozycję X i Y w dwuwymiarowym oknie. Jednak piksele macierzy mają tylko jeden wymiar, przechowujący wartości kolorów w sekwencji liniowej. Patrz rysunek



Weź następujący przykład. Ten szkic ustawia każdy piksel w oknie na losową wartość skali szarości. Tablica pikseli jest podobna do innej tablicy, jedyną różnicą jest to, że nie musimy jej deklarować, ponieważ jest to zmienna wbudowana w Processing.

Przykład 15-5: Ustawianie pikseli

```
size(200,200);

// Before we deal with pixels

loadPixels();

// Loop through every pixel

for (int i = 0; i < pixels.length; i + + ) {    //Możemy uzyskać długość tablicy pikseli, tak jak w przypadku
                                                 //dowolnej tablicy.

// Pick a random number, 0 to 255

float rand = random(255);

// Create a grayscale color based on random number

color c = color(rand);

// Set pixel at that location to random color

pixels[i] = c;

}                    //Możemy uzyskać dostęp do poszczególnych elementów tablicy pikseli za pomocą
                     //indeksu, tak jak w przypadku każdej innej tablicy

// When we are finished dealing with pixels

updatePixels();
```

Po pierwsze, powinniśmy zwrócić uwagę na coś ważnego w powyższym przykładzie. Za każdym razem, gdy uzyskasz dostęp do pikseli okna Przetwarzanie, musisz ostrzec Przetwarzanie do tego działania. Osiąga się to za pomocą dwóch funkcji:

• loadPixels () - Ta funkcja jest wywoływana przed uzyskaniem dostępu do tablicy pikseli, mówiąc: „ładuj piksele, chciałbym z nimi porozmawiać! ”

• updatePixels () - Ta funkcja jest wywoływana po zakończeniu z tablicą pikseli, mówiąc: „Śmiało i zaktualizuj piksele, wszystko gotowe! ”

W przykładzie 15-5, ponieważ kolory są ustawione losowo, nie musieliśmy się martwić o to, gdzie piksele są wyświetlane na ekranie, ponieważ mamy do nich dostęp, ponieważ po prostu ustawiamy wszystkie piksele bez względu na ich względne położenie. Jednak w wielu aplikacjach przetwarzania obrazu lokalizacja XY samych pikseli jest kluczową informacją. Prostym przykładem może być ustawienie każdej parzystej kolumny pikseli na biały i każdy dziwny na czarny. Jak można to zrobić za pomocą jednowymiarowej macierzy pikseli? Skąd wiesz, w której kolumnie lub wierszu znajduje się dany piksel? Programując z pikselami, musimy być w stanie myśleć o każdym pikselu jako o życiu w świecie dwuwymiarowym, ale nadal uzyskiwać dostęp do danych w jednym wymiarze (ponieważ tak jest udostępniany nam). Możemy to zrobić za pomocą następującej formuły:

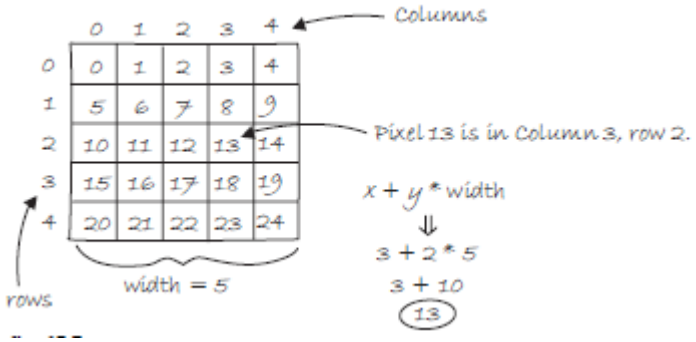
1. Załóż okno lub obraz o danej WIDTH i HEIGHT.

2. Następnie wiemy, że tablica pikseli ma całkowitą liczbę elementów równą SZEROKOŚCI * WYSOKOŚCI.

3. Dla dowolnego punktu X, Y w oknie, lokalizacja w naszej jednowymiarowej macierzy pikseli to:

LOKALIZACJA = X + Y * SZEROKOŚĆ

Może to przypominać dwuwymiarowe tablice w części 13. W rzeczywistości będziemy musieli użyć tej samej techniki zagnieżdżonej dla pętli. Różnica polega na tym, że chociaż chcemy wykorzystać pętle do myślenia o pikselach w dwóch wymiarach, kiedy przechodzimy do rzeczywistego dostępu do pikseli, żyją one w tablicy jednowymiarowej i musimy zastosować formułę z rysunku



Spójrzmy, jak to się robi, wypełniając problem kolumny parzystej / nieparzystej.

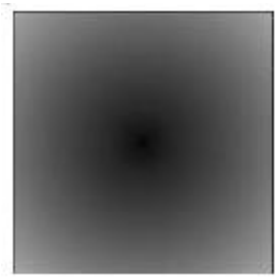
Przykład 15-6: Ustawianie pikseli zgodnie z ich lokalizacją 2D

```
size(200,200);

loadPixels();

// Loop through every pixel column      //Dwie pętle pozwalają nam odwiedzać każdą kolumnę (x) i
                                        //każdy wiersz (y).

for (int x = 0; x < width; x + + ) {

// Loop through every pixel row

for (int y = 0; y < height; y + + ) {

// Use the formula to find the 1D location

int loc = x + y * width;    //Lokalizacja w tablicy pikseli jest obliczana za pomocą naszego wzoru: 1 piksel
                            //lokalizacji = szerokość x + y *
```

```
if (x % 2 = = 0) { // If we are an even column
pixels[loc] = color(255);
} else { // If we are an odd column
pixels[loc] = color(0);
}          //Używamy numeru kolumny (x), aby określić, czy kolor powinien być czarny czy biały.
}
}
updatePixels();
```

Ćwiczenie 15-6: Uzupełnij kod, aby dopasować odpowiednie zrzuty ekranu

```
size(255,255);
_____;
for (int x = 0; x < width; x + + ) {
for (int y = 0; y < height; y + + ) {
int loc = _____;
float distance = _____);
pixels[loc] = _____;
}
}
_____;
```



```
size(255,255);
_____;
for (int x = 0; x < width; x + + ) {
for (int y = 0; y < height; y + + ) {
_____;
if (_____) {
```

_____;

} else {

_____;

}

}

}

_____;

Wprowadzenie do przetwarzania obrazu

W poprzedniej części przyjrzano się przykładom, które ustawiają wartości pikseli zgodnie z dowolnymi obliczeniami. Zobaczymy, jak możemy ustawić piksele zgodnie z tymi, które znajdują się w istniejącym obiekcie PImage. Tutaj jest jakiś dziwny kod.

1. Załaduj plik obrazu do obiektu PImage.

2. Dla każdego piksela w obrazie PImage pobierz kolor piksela i ustaw piksel wyświetlania na ten kolor.

Klasa PImage zawiera przydatne pola, które przechowują dane związane z obrazem - szerokość, wysokość i piksele. Podobnie jak w przypadku klas zdefiniowanych przez użytkownika, możemy uzyskać dostęp do tych pól za pomocą składni kropki.

PImage img = createImage(320,240,RGB); // Make a PImage object

println(img.width); // Yields 320

println(img.height); // Yields 240

img.pixels[0] = color(255,0,0); // Sets the first pixel of the image to red

Dostęp do tych pól pozwala nam przechodzić przez wszystkie piksele obrazu i wyświetlać je na ekranie.

Przykład 15-7: Wyświetlanie pikseli obrazu

PImage img;

void setup() {

size(200,200);

img = loadImage("sunflower.jpg");

}

```
void draw() {

loadPixels();

// Since we are going to access the image's pixels too

img.loadPixels();       //Musimy także wywołać loadPixels () w PImage, ponieważ będziemy czytać
                        //jego piksele.

for (int y = 0; y < height; y + + ) {

for (int x = 0; x < width; x + + ) {

int loc = x + y*width;

// Image Processing Algorithm would go here

float r = red (img.pixels [loc]);       //Funkcje czerwony (), zielony () i niebieski () wyciągają trzy
                                        //składowe koloru z piksela.

float g = green(img.pixels[loc]);

float b = blue (imq.pixels[loc];

// Image Processing would go here       //Gdybyśmy zmienili wartości RGB, zrobilibyśmy to
                                        //tutaj, przed ustawieniem piksela w oknie
                                        //wyświetlacza.

// Set the display pixel to the image pixel

pixels[loc] = color(r,g,b);

}

}

updatePixels();

}
```

Teraz moglibyśmy wymyślić uproszczenia, aby po prostu wyświetlić obraz (np. Pętla zagnieżdżona nie jest wymagana, nie wspominając, że użycie funkcji image () pozwoliłoby nam pominąć cały ten piksel w całości). Jednak przykład 15-7 zapewnia podstawową strukturę uzyskiwania wartości czerwonych, zielonych i niebieskich dla każdego piksela na podstawie jego orientacji przestrzennej (lokalizacja XY); ostatecznie pozwoli to nam opracować bardziej zaawansowane algorytmy przetwarzania obrazu. Zanim przejdziemy dalej, powinienem podkreślić, że ten przykład działa, ponieważ obszar wyświetlania ma takie same wymiary jak obraz źródłowy. Gdyby tak nie było, wystarczy mieć dwa obliczenia lokalizacji pikseli, jeden dla obrazu źródłowego i jeden dla obszaru wyświetlania.

```
int imageLoc = x + y * img.width;

int displayLoc = x + y * szerokość;
```

Ćwiczenie 15-7: Używając przykładu 15-7, zmień wartości r, g i b przed wyświetleniem ich.

Nasz drugi filtr przetwarzania obrazu, tworzenie własnego odcienia () Zaledwie kilka akapitów temu cieszyliśmy się relaksującą sesją kodowania, kolorowania obrazów i dodawania przezroczystości alfa za pomocą przyjaznej metody tint (). W przypadku podstawowego filtrowania ta metoda zadziałała.

Metoda piksel po pikselu pozwoli nam jednak opracować niestandardowe algorytmy do matematycznej zmiany kolorów obrazu. Rozważ jasność - jaśniejsze kolory mają wyższe wartości dla składników czerwonego, zielonego i niebieskiego. Naturalnie wynika z tego, że możemy zmienić jasność obrazu, zwiększając lub zmniejszając składowe koloru każdego piksela. W następnym przykładzie dynamicznie zwiększamy lub zmniejszamy te wartości na podstawie położenia poziomego myszy. (Zauważ, że następne dwa przykłady obejmują tylko obraz sama pętla przetwarzania, zakłada się resztę kodu.)

Przykład 15-8: Regulacja jasności obrazu

```
for (int x = 0; x < img.width; x + + ) {

for (int y = 0; y < img.height; y + + ) {

// Calculate the 1D pixel location

int loc = x + y*img.width;

// Get the R,G,B values from image

float r = red (img.pixels[loc]);

float g = green (img.pixels[loc]);

float b = blue (img.pixels[loc]);

// Change brightness according to the mouse here

float adjustBrightness = ((float) mouseX / width) * 8.0; //Obliczamy mnożnik od 0,0 do 8,0 w oparciu o
                                                          //pozycję mouseX. Ten mnożnik zmienia
                                                          //wartość RGB każdego piksela.

r * = adjustBrightness;

g * = adjustBrightness;

b * = adjustBrightness;

// Constrain RGB to between 0-255

r = constrain(r,0,255);   //Wartości RGB są ograniczone od 0 do 255 przed ustawieniem jako nowy
                          //kolor

g = constrain(g,0,255);

b = constrain(b,0,255);

// Make a new color and set pixel in the window

color c = color(r,g,b);

pixels[loc] = c;

}

}
```

Ponieważ zmieniamy obraz w przeliczeniu na piksel, wszystkie piksele nie muszą być traktowane jednakowo. Na przykład możemy zmienić jasność każdego piksela w zależności od jego odległości od myszy.

Przykład 15-9: Dostosowywanie jasności obrazu na podstawie lokalizacji pikseli

```
for (int x = 0; x < img.width; x + + ) {

for (int y = 0; y < img.height; y + + ) {

// Calculate the 1D pixel location

int loc = x + y*img.width;

// Get the R,G,B values from image

float r = red (img.pixels[loc]);

float g = green (img.pixels[loc]);

float b = blue (img.pixels[loc]);

// Calculate an amount to change brightness

// based on proximity to the mouse

float distance = dist(x,y,mouseX,mouseY);

float adjustBrightness = (50-distance)/50;     //Im bliżej piksela znajduje się mysz, tym niższa jest
                                               //wartość „odległości”. Chcemy, aby bliższe piksele
                                               //były jaśniejsze, więc odwracamy wartość za pomocą
                                               //wzoru:

adjustBrightness (50-odległość) / 50
```

Piksele o odległości 50 (lub większej) mają jasność 0,0 (lub ujemną, co jest równoważne 0), a piksele z odległości 0 mają jasność 1,0.

```
r * = adjustBrightness;

g * = adjustBrightness;

b * = adjustBrightness;

// Constrain RGB to between 0-255

r = constrain(r,0,255);

g = constrain(g,0,255);

b = constrain(b,o,255);

// Make a new color and set pixel in the window

color c = color(r,g,b);

pixels[loc] = c;

}
```

}

Ćwiczenie 15-8: Dostosuj jasność składowych koloru czerwonego, zielonego i niebieskiego oddzielnie w zależności od interakcji myszy. Na przykład, pozwól mouseX kontrolować czerwony, mouseY zielony, odległość niebieski i tak dalej.

Pisanie do innych pikseli obiektu PImage

Wszystkie nasze przykłady przetwarzania obrazu odczytały każdy piksel z obrazu źródłowego i bezpośrednio zapisały nowy piksel w oknie Przetwarzanie. Jednak często wygodniejsze jest zapisywanie nowych pikseli na obrazie docelowym (który następnie wyświetla się za pomocą funkcji image ()). Pokażemy tę technikę, patrząc na inną prostą operację pikseli: próg. Filtr progowy wyświetla każdy piksel obrazu tylko w jednym z dwóch stanów, czarnym lub białym. Stan w stanie jest ustawiany zgodnie z określoną wartością progową. Jeśli jasność piksela jest większa niż próg, kolorujemy piksel biały, mniejszy niż czarny. Przykład 15-10 wykorzystuje arbitralny próg 100.

Przykład 15-10: Próg jasności

```
PImage source; // Source image          //Potrzebujemy dwóch obrazów, obrazu źródłowego
                                        //(oryginalny plik) i docelowego (do wyświetlenia).

PImage destination; // Destination image

void setup() {

size(200,200);

source = loadImage( " sunflower.jpg " );

destination = createImage(source.width, source.height, RGB);

}                    //Obraz docelowy jest tworzony jako pusty obraz o tym samym rozmiarze co źródło.

void draw() {

float threshold = 127;

// We are going to look at both image's pixels

source.loadPixels();

destination.loadPixels();

for (int x = 0; x < source.width; x + + ) {

for (int y = 0; y < source.height; y + + ) {

int loc = x + y*source.width;

// Test the brightness against the threshold //brightness () zwraca wartość między 0 a 255, ogólną
                                                //jasność koloru piksela. Jeśli jest więcej niż 100, zmień
                                                //kolor na biały mniej niż 100, zrób to czarny.

if (brightness(source.pixels[loc]) > threshold){

destination.pixels[loc] = color(255); // White

} else {
```

```
destination.pixels[loc] = color(0); // Black //Zapis do pikseli obrazu docelowego.

}

}

}

// We changed the pixels in destination

destination.updatePixels();

// Display the destination

image(destination,0,0); //Musimy wyświetlić obraz docelowy!

}
```

Ćwiczenie 15-9: Zwiąż próg z lokalizacją myszy.

Jest to szczególna funkcjonalność dostępna bez przetwarzania na piksel w ramach funkcji filtrowania () przetwarzania. Zrozumienie kodu niższego poziomu jest jednak kluczowe, jeśli chcesz zaimplementować własne algorytmy przetwarzania obrazu, niedostępne z fi lterami (). Jeśli wszystko, co chcesz zrobić, to próg, przykład 15-11 jest znacznie prostszy.

Przykład 15-11: Próg jasności z filtrem

```
// Draw the image

image(img,0,0);

// Filter the window with a threshold effect

// 0.5 means threshold is 50% brightness

filter(THRESHOLD,0.5);
```

Więcej o filter():

```
filter(tryb);

filter(tryb, poziom);
```

Funkcja filter() oferuje zestaw wstępnie zapakowanych filtrów dla okna wyświetlania. Nie jest konieczne używanie PImage, filtr zmieni wygląd tego, co jest narysowane w oknie w momencie jego wykonania. Inne dostępne tryby oprócz THRESHOLD to SZARY, INVERT, POSTERIZE, BLUR, OPAQUE, ERODE i DILATE.

Poziom II: Przetwarzanie grupowe pikseli

W poprzednich przykładach widzieliśmy relację jeden do jednego między pikselami źródłowymi a pikselami docelowymi. Aby zwiększyć jasność obrazu, pobieramy jeden piksel z obrazu źródłowego, zwiększamy wartości RGB i wyświetlamy jeden piksel w oknie wyjściowym. Aby wykonać bardziej zaawansowane funkcje przetwarzania obrazu, musimy jednak przejść od paradygmatu od jednego do jednego do przetwarzania grup pikseli. Zacznijmy od utworzenia nowego piksela z dwóch pikseli z obrazu źródłowego - piksela i jego sąsiada po lewej stronie. Jeśli wiemy, że piksel znajduje się w (x, y):

```
int loc = x + y * img.width;
```

color pix = img.pixels [loc];

Następnie jego lewy sąsiad znajduje się przy (x - 1, y):

int leftLoc = (x-1) + y * img.width;

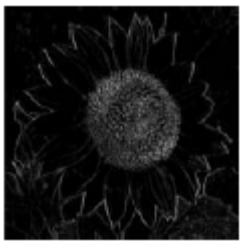
color leftPix = img.pixels[leftLoc];

Moglibyśmy wtedy stworzyć nowy kolor z różnicy między pikselem a jego sąsiadem po lewej stronie.

float diff = abs (jasność (pix) - jasność (leftPix));

pixels [loc] = color (diff);

Przykład 15-12 przedstawia pełny algorytm z wynikami pokazanymi na rysunku

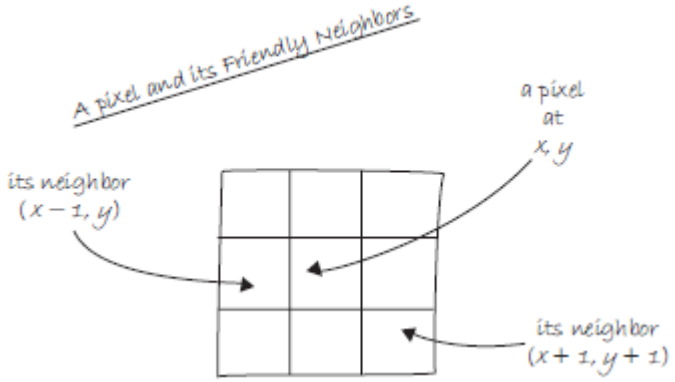


Przykład 15-12: Różnice sąsiadów pikseli (krawędzie)

```
// Since we are looking at left neighbors
// We skip the first column
for (int x = 1; x < width; x + + ) {
for (int y = 0; y < height; y + + ) {
// Pixel location and color
int loc = x + y*img.width;
color pix = img.pixels[loc];
// Pixel to the left location and color
int leftLoc = (x –1) + y*img.width;
color leftPix = img.pixels[leftLoc];
// New color is difference between pixel and left neighbor
float diff = abs(brightness(pix) - brightness(leftPix));
pixels[loc] = color(diff);
}
}
```

Przykład 15-12 to prosty algorytm wykrywania krawędzi pionowych. Gdy piksele znacznie się różnią od sąsiadów, są to najprawdopodobniej piksele „krawędziowe”. Na przykład wyobraź sobie zdjęcie białego kawałka papieru na czarnym stole. Krawędzie tego papieru są tam, gdzie kolory są najbardziej

różne, gdzie biały spotyka się z czernią. W przykładzie 15-12 przyglądamy się dwóm pikselom, aby znaleźć krawędzie. Bardziej wyrafinowane algorytmy zazwyczaj jednak wymagają spojrzenia na wiele innych sąsiadujących pikseli. W końcu każdy piksel ma ośmiu bezpośrednich sąsiadów: górny lewy, górny, górny prawy, prawy, dolny prawy, dolny, dolny lewy i lewy. Patrz rysunek



Te algorytmy przetwarzania obrazu są często określane jako „splot przestrzenny. „Proces wykorzystuje średnią ważoną piksela wejściowego i jego sąsiadów do obliczenia piksela wyjściowego. Innymi słowy, ten nowy piksel jest funkcją obszaru pikseli. Można zastosować sąsiednie obszary o różnych rozmiarach, takie jak macierz 3 x 3, 5 x 5 i tak dalej. Różne kombinacje wag dla każdego piksela dają różne efekty. Na przykład „wyostrzamy" obraz, odejmując wartości sąsiednich pikseli i zwiększając piksel punktu środkowego. Rozmycie osiąga się, biorąc średnią wszystkich sąsiadujących pikseli. (Zauważ, że wartości w macierzy splotu sumują się do 1.)

Na przykład,

Wyostrzenie:

- 1 - 1 - 1

- 1 9 - 1

- 1 - 1 - 1

Rozmycie:

1/9 1/9 1/9

1/9 1/9 1/9

1/9 1/9 1/9

Przykład 15-13 wykonuje splot za pomocą tablicy 2D (patrz rozdział 13, aby przejrzeć tablice 2D), aby zapisać wagi pikseli w macierzy 3 x 3. Ten przykład jest prawdopodobnie najbardziej zaawansowanym przykładem, jaki do tej pory napotkaliśmy w tej książce, ponieważ obejmuje tak wiele elementów (zagnieżdżone pętle, tablice 2D, piksele PImage itp.).

Przykład 15-13: Wyostrz z splotem

```
PImage img;

int w = 80;

// It's possible to perform a convolution
```

```
// the image with different matrices
float[][] matrix = { { – 1, – 1, – 1 },
                     { – 1, 9, –1 },    //Macierz splotowa dla efektu „wyostrzenia” przechowywanego jako
                                        //trójwymiarowa tablica 3x3.
                     { – 1, – 1, –1 } ;
void setup() {
size(200,200);
img = loadImage( "sunflower.jpg");
}
void draw() {
// We're only going to process a portion of the image
// so let's set the whole image as the background first
image(img,0,0);
// Where is the small rectangle we will process
int xstart = constrain(mouseX-w/2,0,img.width);
int ystart = constrain(mouseY-w/2,0,img.height);        //W tym przykładzie przetwarzamy tylko część
                                                        //obrazu - prostokąt 80 x 80 wokół lokalizacji
                                                        //myszy
int xend = constrain(mouseX + w/2,0,img.width);
int yend = constrain(mouseY + w/2,0,img.height);
int matrixsize = 3;
loadPixels();
// Begin our loop for every pixel
for (int x = xstart; x < xend; x + + ) {
for (int y = ystart; y < yend; y + + ) {        //Każda lokalizacja piksela (x, y) zostaje przekazana do funkcji
                                                //zwanej convolution (), która zwraca nową wartość koloru do
                                                //wyświetlenia.
color c = convolution(x,y,matrix,matrixsize,img);
int loc = x + y*img.width;
pixels[loc] = c;
}
}
updatePixels();
```

```
stroke(0);
noFill();
rect(xstart,ystart,w,w);
}
color convolution(int x, int y, float[][] matrix, int matrixsize, PImage img) {
float rtotal = 0.0;
float gtotal = 0.0;
float btotal = 0.0;
int offset = matrixsize / 2;
// Loop through convolution matrix
for (int i = 0; i < matrixsize; i + + ) {
for (int j = 0; j < matrixsize; j + + ) {
// What pixel are we testing
int xloc = x + i-offset;
int yloc = y + j-offset;
int loc = xloc + img.width*yloc;
// Make sure we haven't walked off the edge of the pixel array
loc = constrain(loc,0,img.pixels.length-1);
// Calculate the convolution     //Często dobrze jest patrzeć na sąsiednie piksele, aby się upewnić
                                 //przez przypadek nie zjechaliśmy z krawędzi tablicy pikseli.
rtotal += (red(img.pixels[loc]) * matrix[i][j]);
gtotal += (green(img.pixels[loc]) * matrix[i][j]);
btotal += (blue(img.pixels[loc]) * matrix[i][j]);
}                    //Sumujemy wszystkie sąsiednie piksele pomnożone przez wartości w macierzy splotu.
}
// Make sure RGB is within range
rtotal = constrain(rtotal,0,255);
gtotal = constrain(gtotal,0,255);
btotal = constrain(btotal,0,255);
// Return the resulting color
return color(rtotal,gtotal,btotal); //Po ograniczeniu sum w zakresie 0–255 tworzony jest nowy kolor i
                                    //zwracany
```

}

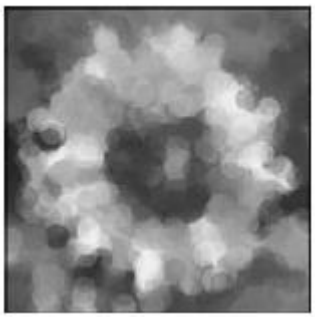
}

Ćwiczenie 15-10: Wypróbuj różne wartości macierzy splotu.

Ćwiczenie 15-11: Korzystając z ram ustanowionych przez nasze przykłady przetwarzania obrazu, utwórz filtr, który pobiera dwa obrazy jako dane wejściowe i generuje jeden obraz wyjściowy. Innymi słowy, każdy wyświetlany piksel powinien być funkcją wartości kolorów z dwóch pikseli, jednego z jednego obrazu i jednego z drugiego. Na przykład, czy możesz napisać kod, aby połączyć dwa obrazy razem (bez użycia tint ())?

Kreatywna wizualizacja

Być może myślisz: „Rany, to wszystko jest bardzo interesujące, ale poważnie, kiedy chcę rozmazać obraz lub zmienić jego jasność, czy naprawdę muszę pisać kod? Czy nie mogę używać Photoshopa? „Rzeczywiście, co mamy Osiągnięty tutaj jest jedynie wstępnym zrozumieniem tego, co robią wysoko wykwalifikowani programiści w Adobe. Siła przetwarzania to jednak potencjał interaktywnych aplikacji graficznych działających w czasie rzeczywistym. Nie musimy żyć w ramach przetwarzania „punktu piksela" i przetwarzania „grupy pikseli". Poniżej przedstawiono dwa przykłady algorytmów do rysowania kształtów. Zamiast kolorowania kształtów losowo lub za pomocą zakodowanych wartości, jak w przeszłości, wybieramy kolory z pikseli obiektu PImage. Sam obraz nigdy nie jest wyświetlany; służy raczej jako baza danych informacji, które możemy wykorzystać do własnych celów twórczych. W tym pierwszym przykładzie, dla każdego cyklu poprzez draw (), wypełniamy jedną elipsę w losowym miejscu na ekranie kolorem pobranym z odpowiedniej lokalizacji w obrazie źródłowym. Rezultatem jest „efekt podobny do punktualistycznego. Patrz rysunek



Przykład 15-14: „Puentylizm"

```
PImage img;

int pointillize = 16;

void setup() {

size(200,200);

img = loadImage( "sunflower.jpg");

background(0);

smooth();

}
```

```
void draw() {

// Pick a random point

int x = int(random(img.width));

int y = int(random(img.height));

int loc = x + y*img.width;

// Look up the RGB color in the source image

loadPixels();

float r = red(img.pixels[loc]);

float g = green(img.pixels[loc]);

float b = blue(img.pixels[loc]);

noStroke();

// Draw an ellipse at that location with that color

fill(r,g,b,100);

ellipse(x,y,pointillize,pointillize);

}       //Wróć do kształtów! Zamiast ustawiania piksela, używamy koloru z piksela, aby narysować
        //okrąg.
```

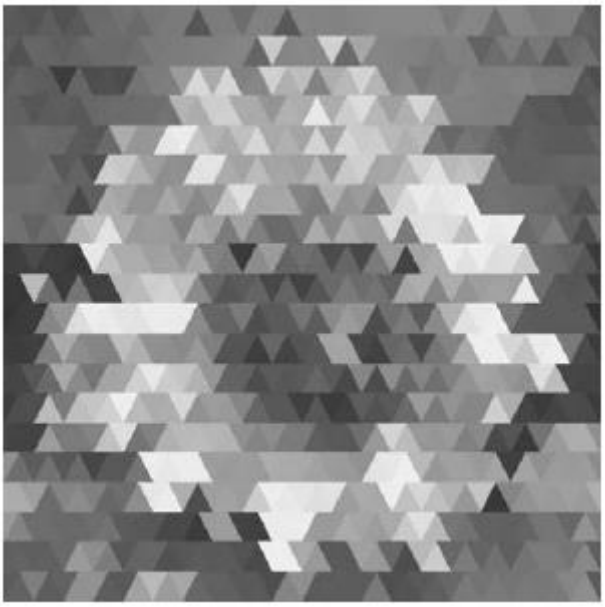
W tym następnym przykładzie pobieramy dane z obrazu dwuwymiarowego i, używając technik translacji 3D opisanych w części 14, renderujemy prostokąt dla każdego piksela w przestrzeni trójwymiarowej. Lokalizacja z zależy od jasności koloru. Jaśniejsze kolory wydają się bliższe widzowi, a ciemniejsze – bliżej

Przykład 15-15: Obraz 2D odwzorowany na 3D

```
PImage img; // The source image

int cellsize = 2; // Dimensions of each cell in the grid

int cols, rows; // Number of columns and rows in our system

void setup() {

size(200,200,P3D);

img = loadImage( " sunflower.jpg " ); // Load the image

cols = width/cellsize; // Calculate # of columns

rows = height/cellsize; // Calculate # of rows

}

void draw() {

background(0);
```

```
loadPixels();

// Begin loop for columns

for (int i = 0; i < cols; i + + ) {

// Begin loop for rows

for (int j = 0; j < rows; j + + ) {

int x = i*cellsize + cellsize/2; // x position

int y = j*cellsize + cellsize/2; // y position

int loc = x + y*width; // Pixel array location

color c = img.pixels[loc]; // Grab the color

// Calculate a z position as a function of mouseX and pixel brightness

float z = (mouseX/(float)width) * brightness(img.pixels[loc])- 100.0;

// Translate to the location, set fill and stroke, and draw the rect

pushMatrix();

translate(x,y,z); //Wartość z dla tłumaczenia 3D jest obliczana jako funkcja jasności piksela, jak również
                  //położenia myszy.

fill(c);

noStroke();

rectMode(CENTER);

rect(0,0,cellsize,cellsize);

popMatrix();

}

}

}
```

Ćwiczenie 15-12: Utwórz szkic, który używa kształtów, aby wyświetlić wzór pokrywający całe okno. Załaduj obraz i pokoloruj kształty zgodnie z pikselami tego obrazu. Na przykład poniższy obraz używa trójkątów



VIII. Video

16.1 Przed Processing

Teraz, gdy zbadaliśmy statyczne obrazy w przetwarzaniu, jesteśmy gotowi przejść do ruchomych obrazów, w szczególności z kamery na żywo (a później z nagranego filmu). Podłączona cyfrowa kamera wideo na PC lub Mac wymaga kilku prostych kroków, zanim będzie można napisać kod.

Na komputerze Mac:

• Podłącz aparat do komputera.

• Upewnij się, że zainstalowano niezbędne oprogramowanie i sterowniki kamery.

• Przetestuj aparat w innym programie, takim jak iChat, aby upewnić się, że działa.

Na komputerze:

• Podłącz aparat do komputera.

• Upewnij się, że zainstalowano niezbędne oprogramowanie i sterowniki kamery.

• Przetestuj aparat. (Użyj dowolnego oprogramowania dostarczonego z aparatem, które umożliwia wyświetlanie strumienia wideo).

• Zainstaluj QuickTime (wersja 7 lub nowsza). Jeśli używasz poprzedniej wersji QuickTime, musisz upewnić się, że wybrałeś instalację „niestandardową" i wybrałeś „QuickTime for Java. "

• Musisz także zainstalować vdig (digitizer wideo), który umożliwia aplikacjom QuickTime (to, co tworzymy) przechwytywanie wideo w systemie Windows. Sytuacja vdig ulega zmianie, więc polecam sprawdzenie strona internetowa tej książki zawierająca aktualizacje. Chociaż nie jest już rozwijany, dostępny jest darmowy vdig ta strona: http://eden.net.nz/7/20071008/. Możesz również rozważyć użycie vdig Abstract Plane (http://www.abstractplane.com.au/products/vdig.jsp), która kosztuje niewielką opłatę, ale ma bardziej zaawansowane funkcje. Wprawdzie proces ten jest nieco mniej skomplikowany na Macu. Dzieje się tak, ponieważ przetwarzanie używa Biblioteki QuickTime w Javie do obsługi wideo. Na komputerach Apple QuickTime jest preinstalowany (chociaż czasami aktualizacje oprogramowania mogą powodować problemy), podczas gdy w systemie Windows musimy upewnić się, że podjęliśmy kroki w celu zainstalowania i prawidłowego skonfigurowania QuickTime. Pod koniec tej części zobaczymy, że w Przetwarzaniu dostępnych jest kilka bibliotek innych firm, które nie wymagają QuickTime do przechwytywania wideo w systemie Windows.

Ćwiczenie 16-1: Podłącz kamerę do komputera. Czy działa w innym programie (nie w przetwarzaniu)? Jeśli jesteś na komputerze, zainstaluj vdig i test.

16.2 Wideo na żywo

Po upewnieniu się, że masz działającą kamerę podłączoną do urządzenia, możesz zacząć pisać kod przetwarzania, aby przechwycić i wyświetlić obraz. Zaczynamy od podstawowych kroków importowania biblioteki wideo i używania klasy Capture do wyświetlania wideo na żywo.

Krok 1. Importuj bibliotekę wideo przetwarzania.

Jeśli pominąłeś rozdział 12 dotyczący przetwarzania bibliotek, możesz wrócić i przejrzeć szczegóły. Nie ma to jednak zbyt wiele, ponieważ biblioteka wideo jest dostarczana z aplikacją Processing.

Wszystko, co musisz zrobić, to zaimportować bibliotekę. Odbywa się to poprzez wybranie opcji menu Szkic → Importuj bibliotekę → wideo lub wpisując następujący wiersz kodu (który powinien znajdować się na samej górze twojego naszkicować):

import processing.video. *;

Użycie opcji menu „Importuj bibliotekę" nie robi nic innego, jak automatyczne wstawienie tej linii do kodu, więc ręczne wpisywanie jest całkowicie równoważne.

Krok 2. Zadeklaruj obiekt przechwytywania

W rozdziale 12 dowiedzieliśmy się, jak tworzyć obiekty z klas wbudowanych w język przetwarzania. Na przykład zrobiliśmy obiekty PImage z klasy Processing PImage. Należy zauważyć, że PImage jest częścią

biblioteka processing.core, a zatem nie była wymagana żadna instrukcja importowa. Biblioteka przetwarzania.video zawiera dwie przydatne klasy - Przechwytywanie, dla wideo na żywo i Film, dla nagranego wideo. Zaczniemy od zadeklarowania obiektu przechwytywania.

Capture video;

Krok 3. Zainicjuj obiekt Capture.

Obiekt wideo „Przechwytywanie" jest jak każdy inny obiekt. Jak dowiedzieliśmy się w rozdziale 8, aby skonstruować obiekt, używamy nowego operatora, po którym następuje konstruktor. W przypadku obiektu Capture ten kod zazwyczaj pojawia się w setup (), zakładając, że chcesz rozpocząć przechwytywanie wideo po rozpoczęciu szkicu.

video = new Capture ();

W powyższym wierszu kodu brakuje odpowiednich argumentów dla konstruktora. Pamiętaj, że nie jest to klasa, którą sami napisaliśmy, więc nie możemy wiedzieć, co jest wymagane między nawiasami bez konsultacji z referencją. Odnośnik online dla konstruktora Capture można znaleźć na stronie internetowej Processing pod adresem:

http://www.processing.org/reference/libraries/video/Capture.html

Odniesienie pokaże, że istnieje kilka sposobów wywołania konstruktora. Typowy sposób wywołania konstruktora Capture to cztery argumenty:

void setup () {

video = new Capture (to, 320,240,30);

}

Przejdźmy przez argumenty użyte w konstruktorze Capture

• this - jeśli jesteś zdezorientowany tym, co to oznacza, nie jesteś sam. Nigdy nie widzieliśmy odniesienia do tego w żadnym z przykładów w tym tekście. Mówiąc technicznie, odnosi się to do instancji klasy, w której pojawia się to słowo. Niestety taka definicja może wywołać wirowanie głowy. Ładniejszym sposobem myślenia o tym jest twierdzenie o autoreferencji. W końcu, co zrobić, jeśli trzeba odwołać się do swojego programu przetwarzającego w ramach własnego kodu? Możesz spróbować powiedzieć „ja" lub „ja". Cóż, te słowa nie są dostępne w Javie, więc zamiast tego mówimy „to. „Powód, dla którego przekazujemy„ this "do obiektu Capture, to:„ Hej, słuchaj, chcę zrobić przechwytywanie wideo, a gdy kamera ma nowy obraz, chcę, żebyś zaalarmował ten aplet. "

• 320 - Na szczęście dla nas, pierwszy argument, to jedyny mylący. 320 odnosi się do szerokości wideo przechwyconego przez kamerę.

• 240 - Wysokość wideo.

• 30 - Pożądana szybkość klatek na sekundę w klatkach na sekundę (fps). To, co wybierzesz, zależy od tego, co robisz. Jeśli zamierzasz na przykład przechwytywać obraz z kamery co jakiś czas, gdy użytkownik kliknie myszką, na przykład, nie będziesz potrzebował dużej liczby klatek na sekundę i może zmniejszyć tę liczbę. Jeśli jednak chcesz wyświetlać wideo na pełnym ekranie na ekranie, wtedy 30 to dobry zakład. Jest to oczywiście po prostu pożądana liczba klatek na sekundę. Jeśli komputer jest zbyt wolny lub żądasz obrazu z kamery o bardzo wysokiej rozdzielczości, wynik może być niższy.

Krok 4. Przeczytaj obraz z kamery.

Istnieją dwie strategie odczytu klatek z kamery. Spójrzmy na oba i nieco arbitralnie wybieramy jedną z pozostałych przykładów w tej części. Obie strategie działają jednak na tej samej podstawowej zasadzie: chcemy tylko odczytać obraz z kamery, gdy dostępna jest nowa ramka do odczytania. Aby sprawdzić, czy obraz jest dostępny, używamy funkcji available (), która zwraca true lub false, w zależności od tego, czy coś tam jest. Jeśli jest, wywoływana jest funkcja read () i ramka z kamery jest odczytywana do pamięci. Robimy to wielokrotnie w pętli draw (), zawsze sprawdzając, czy nowy obraz jest dla nas darmowy.

```
void draw() {

if (video.available()) {

video.read();

}

}
```

Druga strategia, podejście „zdarzenie", wymaga funkcji, która wykonuje się za każdym razem, gdy wystąpi określone zdarzenie, w tym przypadku zdarzenie kamery. Jeśli pamiętasz z części 3, funkcja mousePressed () jest wykonywana za każdym razem, gdy wciśnięty jest przycisk myszy. W przypadku wideo mamy opcję zaimplementowania funkcji captureEvent (), która jest wywoływana za każdym razem, gdy pojawia się zdarzenie przechwytywania, to znaczy nowa kamera jest dostępna z kamery. Te funkcje zdarzeń (mousePressed (), keyPressed (), captureEvent () itd.) Są czasami nazywane „wywołaniem zwrotnym. „A na marginesie, jeśli śledzisz uważnie, to właśnie tu to pasuje. Obiekt Capture,„ wideo ", wie, aby powiadomić ten aplet, wywołując captureEvent (), ponieważ przekazaliśmy mu odniesienie do siebie podczas tworzenia„ wideo . „CaptureEvent () jest funkcją i dlatego musi żyć we własnym bloku, poza setup () i draw ().

```
void captureEvent(Capture video) {

video.read();

}
```

Podsumowując, chcemy wywołać funkcję read () za każdym razem, gdy mamy coś do przeczytania i możemy to zrobić albo przez ręczne sprawdzenie za pomocą komendy available () w draw (), albo zezwolenie na wywołanie zwrotne dla nas - captureEvent () . Wiele innych bibliotek, które omówimy w kolejnych rozdziałach (takich jak sieć i serial), będzie działać dokładnie w ten sam sposób.

Krok 5. Wyświetl obraz wideo.

To bez wątpienia najłatwiejsza część. Możemy myśleć o obiekcie Capture jako PImage, który zmienia się w czasie i w rzeczywistości obiekt Capture można wykorzystać w identyczny sposób jak obiekt PImage.

image(video, 0,0);

Wszystko to zostało zebrane w przykładzie 16-1.

Przykład 16-1: Wyświetl wideo

```
// Step 1. Import the video library
import processing.video.*;        //Krok 1. Importuj bibliotekę wideo
// Step 2. Declare a Capture object
Capture video;  //Krok 2. Zadeklaruj obiekt Capture!
void setup() {
size(320,240);
// Step 3. Initialize Capture object via Constructor
// video is 320 x 240, @15 fps
video = new Capture(this,320,240,15);
}
//Krok 3. Inicjalizuj obiekt Capture! To rozpoczyna proces przechwytywania
void draw() {
// Check to see if a new frame is available
if (video.available()) {
// If so, read it.
video.read();     //Krok 4. Przeczytaj obraz z kamery
}
// Display the video image
image(video,0,0);          //Krok 5. Wyświetl obraz
}
```

Ponownie, wszystko, co możemy zrobić z PImage (zmiana rozmiaru, zabarwienie, przesunięcie itp.) Możemy zrobić za pomocą obiektu Capture. Tak długo, jak czytamy () z tego obiektu, obraz wideo będzie aktualizowany podczas manipulowania nim. Patrz przykład 16-2.

Przykład 16-2: Manipuluj obrazem wideo

```
// Step 1. Import the video library
import processing.video.*;
```

```
Capture video;

void setup() {

size(320,240);

video = new Capture(this,320,240,15);

}

void draw() {

if (video.available()) {

video.read();

}

// Tinting using mouse location

tint(mouseX,mouseY,255);

// Width and height according to mouse

image(video,0,0,mouseX,mouseY); //Obraz wideo może być także przyciemniany i zmieniany tak samo,
                                //jak w przypadku PImage.

}
```

Każdy przykład pojedynczego obrazu z części 15 można odtworzyć za pomocą wideo. Poniżej znajduje się przykład „dostosowywania jasności” (15-19) z obrazem wideo

Przykład 16-3: Dostosuj jasność wideo

```
// Step 1. Import the video library

import processing.video.*;

// Step 2. Declare a Capture object

Capture video;

void setup() {

size(320,240);

// Step 3. Initialize Capture object via Constructor

video = new Capture(this,320,240,15); // video is 320 x 240, @15 fps

background(0);

}

void draw () {

// Sprawdź, czy dostępna jest nowa ramka

if (video.available ()) {

// Jeśli tak, przeczytaj to.
```

```
video.read ();
}
loadPixels ();
video.loadPixels ();
for (int x = 0; x video.width; x + +) {
for (int y = 0; y video.height; y + +) {
// oblicz położenie 1D z siatki 2D
int loc = x + y * video.width;
// pobierz wartości R, G, B z obrazu
float r, g, b;
r = czerwony (video.pixels [loc]);
g = zielony (video.pixels [loc]);
b = niebieski (video.pixels [loc]);
// oblicz kwotę zmiany jasności na podstawie odległości od myszy
float maxdist = 100; // dist (0,0, szerokość, wysokość);
float d = dist (x, y, mouseX, mouseY);
float adjustbrightness = (maxdist-d) / maxdist;
r * = regulacja jasności;
g * = dostosuj jasność;
b * = regulacja jasności;
// ogranicz RGB, aby upewnić się, że mieszczą się w zakresie kolorów 0-255
r = ograniczenie (r, 0,255);
g = ograniczenie (g, 0,255);
b = ograniczenie (b, 0,255);
// wykonaj nowy kolor i ustaw piksel w oknie
kolor c = kolor (r, g, b);
pixels [loc] = c;
}
}
updatePixels ();
}
```

Ćwiczenie 16-2: Odtwórz przykład 15-14 (pointylizm), aby pracować z podglądem na żywo

16.3 Nagrane wideo

Wyświetlanie nagranego wideo ma tę samą strukturę, co wideo na żywo. Przetwarzanie biblioteki wideo akceptuje tylko filmy w formacie QuickTime. Jeśli Twój plik wideo ma inny format, musisz go przekonwertować lub zbadać za pomocą biblioteki innej firmy. Pamiętaj, że odtwarzanie nagranego filmu w systemie Windows nie wymaga vdig.

Krok 1. Zamiast obiektu Capture, zadeklaruj obiekt Movie.

Movie movie;

Krok 2. Inicjalizuj obiekt Movie.

movie = new Movie (this, "testmovie.mov");

Jedynymi niezbędnymi argumentami są ten i nazwa pliku filmu en0closed w cudzysłowach. Plik filmowy powinien być przechowywany w katalogu danych szkicu.

Krok 3. Rozpocznij odtwarzanie filmu.

Tutaj są dwie opcje, play (), która odtwarza film raz, lub loop (), która zapętla go w sposób ciągły.

movie.loop ();

Krok 4. Przeczytaj ramkę z filmu.

Ponownie jest to identyczne z przechwytywaniem. Możemy sprawdzić, czy dostępna jest nowa ramka, lub użyć funkcji zwrotnej.

void draw () {

if (movie.available ()) {

movie.read ();

}

}

Lub:

```
void movieEvent (Film) {
movie.read ();
}
```

Krok 5. Wyświetl film.

```
image(movie,0,0);
```

Przykład 16-4 przedstawia cały program.

Przykład 16-4: Wyświetl film QuickTime

```
import processing.video.*;
Movie movie; // Step 1. Declare Movie object
void setup() {
size(200,200);
// Step 2. Initialize Movie object
movie = new Movie(this, "testmovie.mov"); // Movie file should be in data folder
// Step 3. Start movie playing
movie.loop();
}
// Step 4. Read new frames from movie
void movieEvent(Movie movie) {
movie.read();
}
void draw() {
// Step 5. Display movie.
image(movie,0,0);
}
```

Chociaż Processing w żadnym wypadku nie jest najbardziej wyrafinowanym środowiskiem do wyświetlania i manipulowania nagranymi materiałami wideo (należy zauważyć, że wydajność z dużymi plikami wideo będzie dość powolna), w bibliotece wideo dostępne są bardziej zaawansowane funkcje . Istnieją funkcje umożliwiające uzyskanie czasu trwania filmu (długość mierzona w sekundach), przyspieszenie go i spowolnienie oraz przeskoczenie do określonego punktu w wideo (między innymi). Poniżej znajduje się przykład, który wykorzystuje funkcje jump () (skok do określonego punktu w filmie) i duration () (zwraca długość filmu).

Przykład 16-5: Przewijanie do przodu i do tyłu w filmie

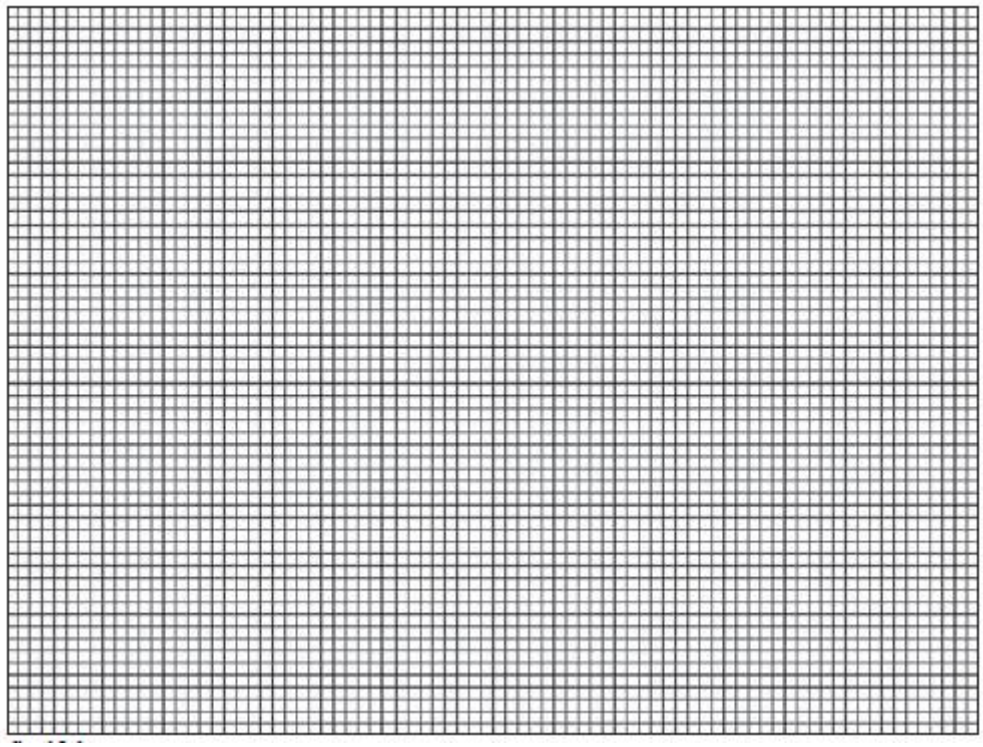
```
// If mouseX is 0, go to beginning
```

```
// If mouseX is width, go to end

// And everything else scrub in between

import processing.video.*;

Movie movie;

void setup() {

size(200,200);

background(0);

movie = new Movie(this, "testmovie.mov");

}

void draw() {

// Ratio of mouse X over width

float ratio = mouseX / (float) width;

// Jump to place in movie based on duration

movie.jump(ratio*movie.duration()); //Funkcja jump() umożliwia natychmiastowe przejście do
                                    //punktu czasowego wideo. duration () zwraca całkowitą
                                    //długość filmu w sekundach.

movie.read(); // read frame

image(movie,0,0); // display frame

}
```

Ćwiczenie 16-3: Używając metody speed () w klasie Movie, napisz program, w którym użytkownik może kontrolować szybkość odtwarzania filmu za pomocą myszy. Uwaga speed () pobiera jeden argument i mnoży szybkość odtwarzania filmu przez tę wartość. Pomnożenie przez 0.5 spowoduje, że film będzie odtwarzany o połowę szybciej, o 2, dwa razy szybciej, o -2, dwa razy szybciej w odwrotnym kierunku i tak dalej.

16.4 Lustra oprogramowania

Dzięki niewielkim kamerom wideo podłączanym do coraz większej liczby komputerów osobistych coraz bardziej popularne staje się opracowywanie oprogramowania, które manipuluje obrazem w czasie rzeczywistym. Te typy aplikacji są czasami określane jako „lustra", ponieważ zapewniają cyfrową refleksję obrazu widza. Rozbudowana biblioteka funkcji przetwarzania grafiki i jej zdolność do przechwytywania z kamery w czasie rzeczywistym sprawiają, że jest to doskonałe środowisko do prototypowania i eksperymentowania z oprogramowaniem lustrzanym. Jak widzieliśmy wcześniej w tym rozdziale, możemy zastosować podstawowe techniki przetwarzania obrazu do obrazów wideo, odczytywać i zastępować piksele jeden po drugim. Idąc o krok dalej, możemy odczytać piksele i zastosować kolory do kształtów narysowanych na ekranie. Zaczniemy od przykładu, który przechwytuje wideo w rozdzielczości 80 x 60 pikseli i renderuje go w oknie 640 x 480. Dla każdego piksela w filmie narysujemy prostokąt o szerokości 8 pikseli i wysokości 8 pikseli. Najpierw napiszmy program, który wyświetla siatkę prostokątów. Patrz rysunek



Przykład 16-6: Rysowanie siatki 8 x 8 pól

```
// Size of each cell in the grid, ratio of window size to video size
int videoScale = 8;          // Zmienna videoScale mówi nam stosunek rozmiaru piksela okna do
                             //rozmiaru sieci.
                             //80 * 8 = 640
                             //60 * 8 = 480
// Number of columns and rows in our system
int cols, rows;
void setup() {
size(640,480);
// Initialize columns and rows
cols = width/videoScale;
rows = height/videoScale;
}
void draw() {
// Begin loop for columns
for (int i = 0; i < cols; i + + ) {
// Begin loop for rows
```

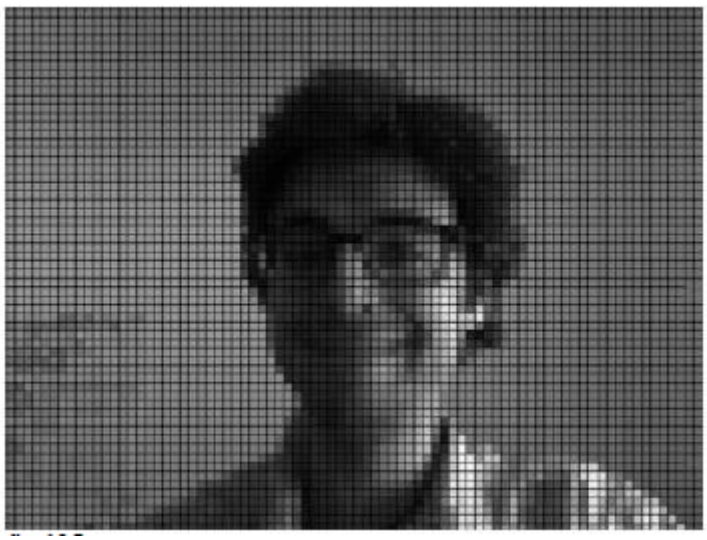
```
for (int j = 0; j <  rows; j + + ) {

// Scaling up to draw a rectangle at (x,y)

int x = i*videoScale;

int y = j*videoScale;

fill(255);

stroke(0);

rect(x,y,videoScale,videoScale); // Dla każdej kolumny i wiersza prostokąt jest rysowany przy (x, y)
                                 //skalowana lokalizacja i wielkość według videoScale.

}

}

}
```

Wiedząc, że chcemy mieć kwadraty o szerokości 8 pikseli i wysokości 8 pikseli, możemy obliczyć liczbę kolumn jako szerokość podzieloną przez osiem i liczbę wierszy jako wysokość podzieloną przez osiem.

• 640/8 = 80 kolumn

• 480/8 = 60 rzędów

Możemy teraz przechwycić obraz wideo o wymiarach 80 x 60. Th jest przydatny, ponieważ przechwytywanie wideo 640 x 480 z kamery może być powolne w porównaniu do 80 x 60. Chcemy tylko przechwycić informacje o kolorze w rozdzielczości wymaganej dla nasz szkic.



Dla każdego kwadratu w kolumnie i i wierszu j sprawdzamy kolor piksela (i, j) w obrazie wideo i odpowiednio go kolorujemy. Patrz przykład 16-7

Przykład 16-7: Pixelacja wideo

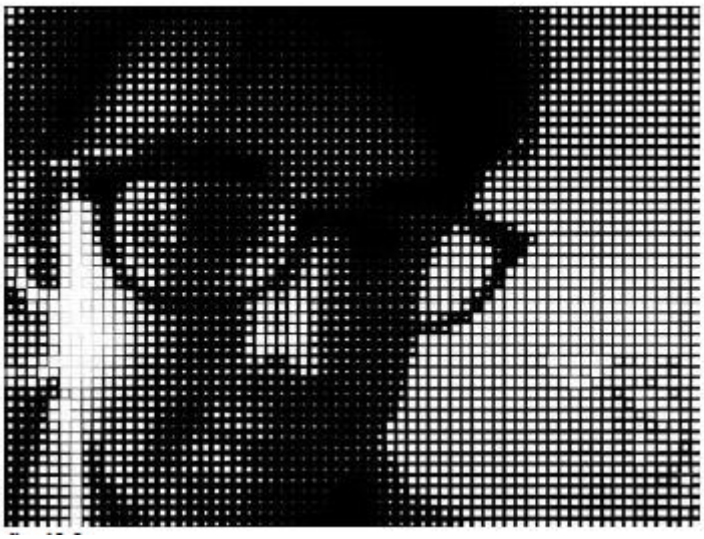
```
// Size of each cell in the grid, ratio of window size to video size

int videoScale = 8;
```

```
// Number of columns and rows in our system
int cols, rows;
// Variable to hold onto Capture object
Capture video;
void setup() {
size(640,480);
// Initialize columns and rows
cols = width/videoScale;
rows = height/videoScale;
background(0);
video = new Capture(this,cols,rows,30);
}
void draw() {
// Read image from the camera
if (video.available()) {
video.read();
}
video.loadPixels();
// Begin loop for columns
for (int i = 0; i    cols; i + + ) {
// Begin loop for rows
for (int j = 0; j    rows; j + + ) {
// Where are we, pixel-wise?
int x = i*videoScale;
int y = j*videoScale;
// Looking up the appropriate color in the pixel array
color c = video.pixels[i + j*video.width];
fill(c);                    // Kolor każdego kwadratu jest pobierany z tablicy pikseli obiektu Capture
stroke(0);
rect(x,y,videoScale,videoScale);
}
```

}

}

Jak widać, rozwinięcie prostego systemu siatki w celu uwzględnienia kolorów z wideo wymaga tylko kilku dodatków. Musimy zadeklarować i zainicjować obiekt Capture, odczytać z niego i pobrać kolory z tablicy pikseli. Można również zastosować mniej dosłowne odwzorowanie kolorów pikseli na kształty w siatce. W poniższym przykładzie używane są tylko kolory czarny i biały. Kwadraty są większe, gdy pojawiają się jaśniejsze piksele w filmie, a mniejsze w przypadku ciemniejszych pikseli. Patrz rysunek



Przykład 16-8: Lustro jasności

```
// Each pixel from the video source is drawn as a
// rectangle with size based on brightness.
import processing.video.*;
// Size of each cell in the grid
int videoScale = 10;
// Number of columns and rows in our system
int cols, rows;
// Variable for capture device
Capture video;
void setup() {
size(640,480);
// Initialize columns and rows
cols = width/videoScale;
rows = height/videoScale;
smooth();
```

```
// Construct the Capture object
video = new Capture(this,cols,rows,15);
}
void draw() {
if (video.available()) {
video.read();
}
background(0);
video.loadPixels();
// Begin loop for columns        // Aby odzwierciedlić obraz, kolumna jest odwracana przy użyciu
                                 //następującego wzoru:
                                 // mirrored column = width – column – 1
for (int i = 0; i    cols; i + + ) {
// Begin loop for rows
for (int j = 0; j    rows; j + + ) {
// Where are we, pixel-wise?
int x = i*videoScale;
int y = j*videoScale;
// Reversing x to mirror the image
int loc = (video.width – i – 1) + j*video.width;
// Each rect is colored white with a size determined by brightness
color c = video.pixels[loc];
float sz = (brightness(c)/255.0)*videoScale;
rectMode(CENTER);             // Rozmiar prostokąta jest obliczany jako funkcja jasności piksela. ZA
                              //jasny piksel jest dużym prostokątem, a ciemny piksel jest małym.
fill(255);
noStroke();
rect(x + videoScale/2,y + videoScale/2,sz,sz);
}
}
}
```

Często warto pomyśleć o opracowaniu oprogramowania lustrzanego w dwóch krokach. Pomoże to również myśleć o bardziej oczywistym odwzorowaniu pikseli na kształty na siatce.

Krok 1. Opracuj ciekawy wzór, który obejmuje całe okno.

Krok 2. Użyj pikseli wideo jako tabeli wyszukiwania do wybarwienia tego wzoru.

Na przykład, powiedzmy dla kroku 1, piszemy program, który pisze losową linię wokół okna. Oto nasz algorytm, napisany pseudokodem.

• Zacznij od xiy jako współrzędnych na środku ekranu.

• Powtarzaj na zawsze:

- Wybierz nowy x i y (pozostając w krawędziach).

- Narysuj linię od starej (x, y) do nowej (x, y).

- Zapisz nowe (x, y).

Przykład 16-9: Scribbler

```
// Two global variables
float x;
float y;
void setup() {
size(320,240);
smooth();
background(255);
// Start x and y in the center
x = width/2;
y = height/2;
}
void draw() {
// Pick a new x and y
float newx = constrain(x + random( –20,20),0,width);   // Nowa lokalizacja x, y jest wybierana jako
                                                       //prąd (x, y) plus lub minus wartość losowa.
                                                       //Nowa lokalizacja jest ograniczona w
                                                       //pikselach okna.
float newy = constrain(y + random( –20,20),0,height);
// Draw a line from x,y to the newx,newy
stroke(0);
strokeWeight(4);
```

```
line(x,y,newx,newy);

x = newx;          // Zapisujemy nową lokalizację w (x, y), aby ponownie rozpocząć proces

y = newy;

}
```

Teraz, kiedy skończyliśmy szkic generowania wzoru, możemy zmienić obrys (), aby ustawić kolor zgodnie z obrazem wideo.

Przykład 16-10: Lusterko bazgrołów

```
import processing.video.*;

// Two global variables

float x;

float y;

// Variable to hold onto Capture object

Capture video;

void setup() {

size(320,240);

smooth();

// framerate(30);

background(0);

// Start x and y in the center

x = width/2;

y = height/2;

// Start the capture process

video = new Capture(this,width,height,15); // Gdyby okno było większe (powiedzmy 800 x 600),
                                           //możemy chcieć wprowadzić zmienną videoScale,
                                           //abyśmy nie musieli rejestrować tak dużego obrazu.

}

void draw() {

// Read image from the camera

if (video.available()) {

video.read();

}

video.loadPixels();
```

```
// Pick a new x and y

float newx = constrain(x + random( - 20,20),0,width-1);

float newy = constrain(y + random( - 20,20),0,height-1);

// Find the midpoint of the line

int midx = int((newx + x) / 2);

int midy = int((newy + y) / 2);

// Pick the color from the video, reversing x

color c = video.pixels[(width-1-midx) + midy*video.width]; // Kolor rysika jest pobierany z piksela na
                                                           //obrazie wideo.

// Draw a line from x,y to the newx,newy

stroke(c);

strokeWeight(4);

line(x,y,newx,newy);

// Save newx, newy in x,y

x = newx;

y = newy;

}
```

Ćwiczenie 16-4: Stwórz własne oprogramowanie lustrzane, korzystając z metodologii z przykładów 16-9 i 16-1 0. Utwórz system bez wideo, a następnie włącz wideo, aby określić kolory, zachowania i tak dalej.

16.5 Wideo jako czujnik, widzenie komputerowe

Każdy przykład w tym rozdziale traktuje kamerę wideo jako źródło danych dla zdjęć cyfrowych wyświetlanych na ekranie. Ta sekcja zawiera proste wprowadzenie do rzeczy, które możesz zrobić za pomocą kamery wideo, gdy nie wyświetlasz obrazu, czyli „wizja komputerowa. „Wizja komputerowa to naukowa dziedzina badań poświęcona maszynom, które widzą, wykorzystując kamerę jako czujnik. Aby lepiej zrozumieć wewnętrzne działanie algorytmów wizji komputerowej, sami napiszemy cały kod na poziomie piksel po pikselu. Aby jednak zbadać te tematy, warto rozważyć pobranie niektórych bibliotek wizyjnych innych producentów, które są dostępne do przetwarzania. Wiele z tych bibliotek ma zaawansowane funkcje wykraczające poza zakres tego rozdziału. Krótki przegląd bibliotek zostanie udostępniony na końcu tej sekcji. Zacznijmy od prostego przykładu. Kamera wideo jest naszym przyjacielem, ponieważ zapewnia mnóstwo informacji. Obraz 320 x 240 to 76 800 pikseli! A gdybyśmy sprowadzili wszystkie te piksele do jednej liczby: ogólna jasność pokoju? Można to osiągnąć za pomocą jednego dolara czujnika światła (lub „fotokomórki”), ale jako ćwiczenie sprawimy, że zrobi to nasza kamera internetowa. W innych przykładach widzieliśmy, że wartość jasności pojedynczego piksela można odzyskać za pomocą funkcji jasności (), która zwraca liczbę punktów flotacji między 0 a 255. Poniższa linia kodu pobiera jasność dla pierwszego piksela na obrazie wideo.

```
float brightness = brightness (video.pixels [0]);
```

Następnie możemy obliczyć ogólną (tj. Średnią) jasność, sumując wszystkie wartości jasności i dzieląc przez całkowitą liczbę pikseli.

```
video.loadPixels();

// Start with a total of 0

float totalBrightness = 0;

// Sum the brightness of each pixel

for (int i = 0; i < video.pixels.length; i + + ) {

color c = video.pixels[i];

totalBrightness + = brightness(c); // Suma wszystkich wartości jasności.

}

// Compute the average

float averageBrightness = totalBrightness / video.pixels.length; // Średnia jasność = całkowita jasność
                                                                 // / całkowita liczba pikseli

// Display the background as average brightness

background(averageBrightness);
```

Zanim zaczniesz energicznie kibicować temu osiągnięciu, podczas gdy ten przykład stanowi doskonałą demonstrację algorytmu analizującego dane dostarczane przez źródło wideo, nie zaczyna wykorzystywać mocy tego, co można „zobaczyć" za pomocą kamery wideo. W końcu obraz wideo to nie tylko zbiór kolorów, ale zbiór kolorów zorientowanych przestrzennie. Opracowując algorytmy przeszukujące piksele i rozpoznające wzorce, możemy zacząć opracowywać bardziej zaawansowane aplikacje do wizji komputerowej. Śledzenie najjaśniejszego koloru to dobry pierwszy krok. Wyobraź sobie ciemny pokój z pojedynczym ruchomym źródłem światła. Dzięki technikom, które poznamy, źródło światła może zastąpić mysz jako formę interakcji. Tak, jesteś na dobrej drodze do grania w Ponga za pomocą latarki. Najpierw zbadamy, jak przeszukać obraz i znaleźć lokalizację x, y najjaśniejszego piksela. Strategia, którą zastosujemy, polega na przejściu przez wszystkie piksele, w poszukiwaniu „rekordu świata" najjaśniejszego piksela (przy użyciu funkcji jasności ()). Początkowo rekord świata będzie przechowywany przez pierwszy piksel. Jak inne piksele pobiją ten rekord, staną się rekordzistą świata. Pod koniec pętli, niezależnie od tego, który piksel jest obecny, aktualny rekordzista otrzymuje nagrodę „Najjaśniejszy piksel obrazu". Oto kod:

```
// The record is 0 when we fi rst start

float worldRecord = 0.0;

// Which pixel will win the prize?

int xRecordHolder = 0;

int yRecordHolder = 0;

for (int x = 0; x < video.width; x + + ) {

for (int y = 0; y < video.height; y + + ) {
```

```
// What is current brightness

int loc = x + y*video.width;

float currentBrightness = brightness(video.pixels[loc]);

if (currentBrightness > worldRecord) {

// Set a new record

worldRecord = currentBrightness;

// This pixel holds the record!

xRecordHolder = x;      // Kiedy znajdziemy nowy najjaśniejszy piksel, musimy zapisać (x, y) położenie
                        //tego piksela w tablicy, abyśmy mogli uzyskać do niego dostęp później.


yRecordHolder = y;

}

}

}
```

Naturalnym rozszerzeniem tego przykładu byłoby śledzenie konkretnego koloru, a nie tylko najjaśniejszego. Na przykład możemy szukać najbardziej „czerwonego” lub najbardziej „niebieskiego” na obrazie wideo. Aby przeprowadzić tego typu analizę, musimy opracować metodologię porównywania kolorów. Stwórzmy dwa kolory, c1 i c2.

```
color c1 = color(255,100,50);

color c2 = color(150,255,0);
```

Kolory można porównywać tylko pod względem składowych czerwonych, zielonych i niebieskich, więc musimy najpierw oddzielić te wartości.

```
float r1 = red(c1);

float g1 = green(c1);

float b1 = blue(c1);

float r2 = red(c2);

float g2 = green(c2);

float b2 = blue(c2);
```

Teraz jesteśmy gotowi porównać kolory. Jedna strategia polega na przyjęciu sumy wartości bezwzględnej różnic. To kęs, ale jest to naprawdę dość proste. Weź r1 minus r2. Ponieważ zależy nam tylko na wielkości różnicy, a nie na tym, czy jest ona dodatnia czy ujemna, należy przyjąć wartość bezwzględną (wersja dodatnia liczby). Zrób to dla zielonego i niebieskiego i dodaj je wszystkie razem.

```
float diff = abs (r1-r2) + abs (g1-g2) + abs (b1-b2);
```

Chociaż jest to całkowicie wystarczające (i szybkie obliczenie), bardziej dokładnym sposobem obliczenia różnicy między kolorami jest przyjęcie „odległości” między kolorami. OK, więc myślisz: „Um,

poważnie? Jak kolor może być daleko lub blisko innego koloru? „Cóż, wiemy, że odległość między dwoma punktami jest obliczana za pomocą twierdzenia Pitagorasa Th. Możemy myśleć o kolorze jako punkcie w przestrzeni trójwymiarowej, tylko zamiast x, y i z, mamy r, g i b. Jeśli dwa kolory są blisko siebie w tej przestrzeni kolorów, są podobne; jeśli są daleko, są różni.

```
float diff = dist (r1, g1, b1, r2, g2, b2);   // Chociaż dokładniejsza, ponieważ funkcja dist () zawiera
                                              //pierwiastek kwadratowy w obliczeniach, jest wolniejsza niż
                                              //wartość bezwzględna metody różnicowej. Jednym ze
                                              //sposobów jest napisanie własnej funkcji odległości kolorów
                                              //bez pierwiastka kwadratowego. colorDistance (r 1-r 2) * (r1-
                                              //r2) + (g1-g2) * (g1-g2) + (b1-b2) * (b1-b2)
```

Na przykład szukanie piksela „najbardziej czerwonego” w obrazie szuka koloru „najbliższego” do czerwonego - (255,0,0). Dostosowując kod śledzenia jasności, aby znaleźć najbliższy piksel do dowolnego koloru (a nie najjaśniejszy), możemy połączyć szkic śledzenia kolorów. W poniższym przykładzie użytkownik może kliknąć myszą na kolor obrazu, który ma być śledzony. W miejscu najbardziej pasującym do tego koloru pojawi się czarne kółko. Patrz rysunek

Przykład 16-11: Proste śledzenie kolorów

```
import processing.video.*;

// Variable for capture device

Capture video;

color trackColor;          // Zmienna dla koloru, którego szukamy.

void setup() {

size(320,240);

video = new Capture(this,width,height,15);

// Start off tracking for red

trackColor = color(255,0,0);

smooth();

}

void draw() {

// Capture and display the video
```

```
if (video.available()) {
video.read();
}
video.loadPixels();
image(video,0,0);
// Closest record, we start with a high number
float worldRecord = 500;          // Zanim zaczniemy wyszukiwanie, „rekord świata” dla najbliższego
                                  //koloru jest ustawiany na wysoką liczbę, która jest łatwa do
                                  //pokonania dla pierwszego piksela.
// XY coordinate of closest color
int closestX = 0;
int closestY = 0;
// Begin loop to walk through every pixel
for (int x = 0; x < video.width; x + + ) {
for (int y = 0; y < video.height; y + + ) {
int loc = x + y*video.width;
// What is current color
color currentColor = video.pixels[loc];
float r1 = red(currentColor);
float g1 = green(currentColor);
float b1 = blue(currentColor);
float r2 = red(trackColor);
float g2 = green(trackColor);
float b2 = blue(trackColor);
// Using euclidean distance to compare colors
float d = dist(r1,g1,b1,r2,g2,b2);          // Używamy funkcji dist () do porównania bieżącego koloru z
                                            //kolor, który śledzimy
// If current color is more similar to tracked color than
// closest color, save current location and current difference
if (d    worldRecord) {
worldRecord = d;
closestX = x;
```

```
closestY = y;

}

}

}

if (worldRecord < 10) {  // Uznajemy tylko znaleziony kolor, jeśli jego odległość koloru jest mniejsza niż
                         //10. Ten próg 10 jest dowolny i można dostosować tę liczbę w zależności od
                         //dokładności wymaganej do śledzenia.

// Draw a circle at the tracked pixel

fill(trackColor);

strokeWeight(4.0);

stroke(0);

ellipse(closestX,closestY,16,16);

}

}

void mousePressed() {

// Save color where the mouse is clicked in trackColor variable

int loc = mouseX + mouseY*video.width;

trackColor = video.pixels[loc];

}
```

Ćwiczenie 16-5: Weź dowolny utworzony wcześniej szkic Processing, który wymaga interakcji z myszą i zastąp mysz myszą śledzeniem kolorów. Stwórz środowisko dla aparatu, które jest proste i o wysokim kontraście. Na przykład skieruj aparat na czarny blat z małym białym obiektem. Kontroluj swój szkic za pomocą lokalizacji obiektu. Pokazany obraz ilustruje przykład, który kontroluje „węża" (przykład 9-9) za pomocą śledzonego paska.

16.6 Usuwanie tła

Porównanie odległości dla koloru okazuje się przydatne także w innych algorytmach widzenia komputerowego, takich jak usuwanie tła. Powiedzmy, że chciałeś pokazać film, w którym tańczysz hula, tylko ty nie chciałeś tańczyć w swoim biurze, gdzie się znajdujesz, ale na plaży z falami rozbijającymi się za tobą. Usuwanie tła to technika, która pozwala usunąć tło obrazu (biuro)i zastąp go dowolnymi pikselami, które lubisz (plaża), pozostawiając nienaruszony pierwszy plan (taniec). Oto nasz algorytm.

• Zapamiętaj obraz tła.

• Sprawdź każdy piksel w bieżącej ramce wideo. Jeśli jest bardzo różny od odpowiedniego piksela w obrazie tła, jest to piksel pierwszego planu. Jeśli nie, jest to piksel tła. Wyświetl tylko piksele na pierwszym planie.

Aby zademonstrować powyższy algorytm, wykonamy odwrotny zielony ekran. Szkic usunie tło z obrazu i zastąpi je zielonymi pikselami. Krok pierwszy to „zapamiętywanie" tła. Tło jest zasadniczo migawką z

wideo. Ponieważ obraz wideo zmienia się w czasie, musimy zapisać kopię klatki wideo w oddzielnym obiekcie PImage.

```
PImage backgroundImage;

void setup () {

backgroundImage = createImage (video.width, video.height, RGB);

}
```

Po utworzeniu backgroundImage jest to pusty obraz o takich samych wymiarach jak wideo. Nie jest to szczególnie użyteczne w tej formie, dlatego musimy skopiować obraz z kamery do obrazu tła, gdy chcemy zapamiętać tło. Zróbmy to po naciśnięciu myszy.

```
void mousePressed () {

// Copying the current frame of video into the backgroundImage object

// Note copy takes 5 arguments:

// The source image

// x,y,width, and height of region to be copied from the source

// x,y,width, and height of copy destination

backgroundImage.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);

backgroundImage.updatePixels();      // copy () umożliwia kopiowanie pikseli z jednego obrazu do
                                     //drugiego. Zauważ, że updatePixels () powinien zostać
                                     //wywołany po skopiowaniu nowych pikseli!

}
```

Po zapisaniu obrazu tła możemy przejść przez wszystkie piksele w bieżącej ramce i porównać je z tłem za pomocą obliczenia odległości. Dla każdego piksela (x, y) używamy następujący kod:

```
int loc = x + y*video.width; // Step 1, what is the 1D pixel location

color fgColor = video.pixels[loc]; // Step 2, what is the foreground color

color bgColor = backgroundImage.pixels[loc]; // Step 3, what is the background color

// Step 4, compare the foreground and background color

float r1 = red(fgColor); float g1 = green(fgColor); float b1 = blue(fgColor);

float r2 = red(bgColor); float g2 = green(bgColor); float b2 = blue(bgColor);

float diff = dist(r1,g1,b1,r2,g2,b2);

// Step 5, Is the foreground color different from the background color

if (diff > threshold) {

// If so, display the foreground color

pixels[loc] = fgColor;
```

```
} else {
// If not, display green
pixels[loc] = color(0,255,0);
}
```

Powyższy kod zakłada zmienną o nazwie „próg. „Im niższy próg, tym łatwiejszy jest piksel na pierwszym planie. Nie musi być bardzo różny od piksela tła. Oto pełny przykład z progiem jako zmienną globalną.

Przykład 16-12: Proste usuwanie tła

```
// Click the mouse to memorize a current background image
import processing.video.*;
// Variable for capture device
Capture video;
// Saved background
PImage backgroundImage;
// How different must a pixel be to be a foreground pixel
float threshold = 20;
void setup() {
size(320,240);
video = new Capture(this, width, height, 30);
// Create an empty image the same size as the video
backgroundImage = createImage(video.width,video.height,RGB);
}
void draw() {
// Capture video
if (video.available()) {
video.read();
}
loadPixels();     // Patrzymy na piksele wideo, zapamiętane piksele backgroundImage, a także na
                  //dostęp do pikseli wyświetlania. Więc musimy loadPixels () dla wszystkich!
video.loadPixels();
backgroundImage.loadPixels();
// Draw the video image on the background
image(video,0,0);
```

```
// Begin loop to walk through every pixel
for (int x = 0; x < video.width; x + + ) {
for (int y = 0; y < video.height; y + + ) {
int loc = x + y*video.width; // Step 1, what is the 1D pixel location
color fgColor = video.pixels[loc]; // Step 2, what is the foreground color
// Step 3, what is the background color
color bgColor = backgroundImage.pixels[loc];
// Step 4, compare the foreground and background color
float r1 = red(fgColor);
float g1 = green(fgColor);
float b1 = blue(fgColor);
float r2 = red(bgColor);
float g2 = green(bgColor);
float b2 = blue(bgColor);
float diff = dist(r1,g1,b1,r2,g2,b2);
// Step 5, Is the foreground color different from the background color
if (diff > threshold) {
// If so, display the foreground color
pixels[loc] = fgColor;
} else {
// If not, display green
pixels[loc] = color(0,255,0); // Moglibyśmy zastąpić piksele tła czymś innym niż kolor zielony!
}
}
}
updatePixels();
}
void mousePressed() {
// Copying the current frame of video into the backgroundImage object
// Note copy takes 5 arguments:
// The source image
```

// x,y,width, and height of region to be copied from the source

// x,y,width, and height of copy destination

backgroundImage.copy(video,0,0,video.width,video.height,0,0,video.width,video.

height);

backgroundImage.updatePixels();

}

Kiedy ostatecznie przejdziesz do uruchomienia tego przykładu, wyjdź z kadru, kliknij myszką, aby zapamiętać tło bez ciebie, a następnie cofnij się do ramki; zobaczysz wynik widoczny na Rysunek

Jeśli ten szkic w ogóle nie działa, sprawdź i zobacz, jakie funkcje „automatyczne" są włączone w aparacie. Na przykład, jeśli aparat jest ustawiony na automatyczną regulację jasności lub balans bieli, masz problem. Nawet jeśli obraz tła zostanie zapamiętany, gdy cały obraz stanie się jaśniejszy lub zmieni odcień, ten szkic będzie sądził, że wszystkie piksele uległy zmianie i dlatego są częścią pierwszego planu! Aby uzyskać najlepsze wyniki, wyłącz wszystkie funkcje automatyczne w aparacie.

Ćwiczenie 16-6: Zamiast zastępować tło zielonymi pikselami, zastąp je innym obrazem. Jakie wartości działają dobrze dla progu i jakie wartości w ogóle nie działają? Spróbuj sterować zmienną progową za pomocą myszy.

16.7 Wykrywanie ruchu

Dzisiaj jest szczęśliwy dzień. Czemu? Ponieważ wszystkie prace, które wykonaliśmy, aby dowiedzieć się, jak usunąć tło z wideo, umożliwiają nam wykrycie ruchu za darmo. W przykładzie usuwania tła zbadaliśmy zależność każdego piksela od zapisanego obrazu tła. Ruch w obrazie wideo występuje, gdy piksel różni się znacznie od tego, co wcześniej był jedną klatką. Innymi słowy, wykrywanie ruchu jest dokładnie tym ten sam algorytm, tylko raz zamiast zapisywać obraz tła, zapisujemy poprzednią klatkę wideo stale! Poniższy przykład jest identyczny z przykładem usuwania tła z tylko jedną ważną zmianą - poprzednia klatka wideo jest zawsze zapisywana za każdym razem, gdy dostępna jest nowa ramka.

// Capture video

if (video.available()) {

// Save previous frame for motion detection!!

prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);

video.read();

}

(Wyświetlane kolory są również zmieniane na czarno-białe, a niektóre nazwy zmiennych są różne, ale są to trywialne zmiany.)

Przykład 16-13: Proste wykrywanie ruchu

```
import processing.video.*;

// Variable for capture device

Capture video;

// Previous Frame

PImage prevFrame;

// How different must a pixel be to be a "motion" pixel

float threshold = 50;

void setup() {

size(320,240);

video = new Capture(this, width, height, 30);

// Create an empty image the same size as the video

prevFrame = createImage(video.width,video.height,RGB);

}

void draw() {

// Capture video

if (video.available()) {

// Save previous frame for motion detection!!

prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);

prevFrame.updatePixels();

video.read();

}

loadPixels();

video.loadPixels();

prevFrame.loadPixels();

// Begin loop to walk through every pixel

for (int x = 0; x < video.width; x + + ) {

for (int y = 0; y < video.height; y + + ) {

int loc = x + y*video.width; // Step 1, what is the 1D pixel location
```

```
color current = video.pixels[loc]; // Step 2, what is the current color

color previous = prevFrame.pixels[loc]; // Step 3, what is the previous color

// Step 4, compare colors (previous vs. current)

float r1 = red(current); float g1 = green(current); float b1 = blue(current);

float r2 = red(previous); float g2 = green(previous); float b2 = blue(previous);

float diff = dist(r1,g1,b1,r2,g2,b2);

// Step 5, How different are the colors?

if (diff > threshold) {

// If motion, display black

pixels[loc] = color(0);

} else {

// If not, display white

pixels[loc] = color(255);

}

}

}

updatePixels();

}
```

Co jeśli chcemy poznać „ogólny” ruch w pokoju? Na początku sekcji 16.5 obliczyliśmy średnią jasność obrazu, biorąc sumę jasności każdego piksela i dzieląc go przez całkowitą liczbę pikseli.

Średnia jasność = całkowita jasność / całkowita liczba pikseli

Możemy obliczyć średni ruch w ten sam sposób:

Średni ruch = całkowity ruch / całkowita liczba pikseli

Poniższy przykład wyświetla okrąg, który zmienia kolor i rozmiar na podstawie średniej ilości ruchu. Pamiętaj, że nie musisz wyświetlać wideo, aby go przeanalizować!

Przykład 16-14: Ruch ogólny

```
import processing.video.*;

// Variable for capture device

Capture video;

// Previous Frame

PImage prevFrame;

// How different must a pixel be to be a "motion" pixel
```

```
float threshold = 50;
void setup() {
size(320,240);
// Using the default capture device
video = new Capture(this, width, height, 15);
// Create an empty image the same size as the video
prevFrame = createImage(video.width,video.height,RGB);
}
void draw() {
background(0);
// If you want to display the videoY
// You don't need to display it to analyze it!
image(video,0,0);
// Capture video
if (video.available()) {
// Save previous frame for motion detection!!
prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height);
prevFrame.updatePixels();
video.read();
}
loadPixels();
video.loadPixels();
prevFrame.loadPixels();
// Begin loop to walk through every pixel
// Start with a total of 0
float totalMotion = 0;
// Sum the brightness of each pixel
for (int i = 0; i < video.pixels.length; i + + ) {
color current = video.pixels[i];
// Step 2, what is the current color
color previous = prevFrame.pixels[i];
```

```
// Step 3, what is the previous color

// Step 4, compare colors (previous vs. current)

float r1 = red(current); float g1 = green(current);

float b1 = blue(current);

float r2 = red(previous); float g2 =

float b2 = blue(previous);

float diff = dist(r1,g1,b1,r2,g2,b2);        // Ruch pojedynczego piksela jest różnicą między poprzednim
                                             //kolorem a bieżącym kolorem.

totalMotion + = diff;      // totalMotion jest sumą wszystkich różnic kolorów

}

float avgMotion = totalMotion / video.pixels.length; // averageMotion to ruch całkowity podzielony
                                                     //przez liczbę analizowanych pikselii

// Draw a circle based on average motion

smooth();

noStroke();

fill(100 + avgMotion*3,100,100);

float r = avgMotion*2;

ellipse(width/2,height/2,r,r);

}
```

Ćwiczenie 16-7: Utwórz szkic, który szuka średniej lokalizacji ruchu. Czy możesz mieć elipsę podążającą za twoją machającą ręką?

16.8 Komputerowe biblioteki wizyjne

Dostępnych jest już kilka bibliotek wizyjnych do przetwarzania (i nieuchronnie będzie ich więcej). Dobrą rzeczą w pisaniu własnego kodu wizyjnego jest to, że możesz kontrolować algorytm wizyjny na najniższym poziomie, wykonując analizę dokładnie dopasowaną do Twoich potrzeb. Zaletą korzystania z biblioteki innej firmy jest to, że odkąd przeprowadzono wiele badań w zakresie rozwiązywania typowych problemów ze wzrokiem komputerowym (wykrywanie krawędzi, plam, ruchu, śledzenia kolorów itp.), Nie trzeba wykonywać wszystkich trudnych czynności pracuj sam! Oto krótki przegląd trzech obecnie dostępnych bibliotek.

JMyron (WebCamXtra) autorstwa Josha Nimoya i in.

http://webcamxtra.sourceforge.net/

Jedną z zalet korzystania z JMyron jest brak konieczności używania vdig w systemie Windows. Zawiera również wiele wbudowanych funkcji do wykonywania niektórych zadań opisanych w tej części: wykrywanie ruchu i śledzenie kolorów. Będzie również szukać grup podobnych pikseli (znanych jako plamy lub globusy).

LibCV autorstwa Karstena Schmidta

http://toxi.co.uk/p5/libcv/

Podobnie jak JMyron, LibCV nie wymaga QuickTime ani WinVDIG na komputerach z systemem Windows. Jednak zamiast korzystać z kodu natywnego, używa Java Media Framework (JMF) do łączenia się z obrazami i przechwytywania ich z cyfrowej kamery wideo. LibCV zawiera także funkcje niedostępne w niektórych innych bibliotekach komputerowych, takich jak „uczenie się w tle, odejmowanie tła, obrazy różnicowe i trapez (korekta perspektywy). ”

BlobDetection Juliena „v3ga” Gachadoat

http://www.v3ga.net/processing/BlobDetection/

Jest to biblioteka, jak sama nazwa wskazuje, jest specjalnie zaprojektowana do wykrywania plam na obrazie. Krople są zdefiniowane jako obszary pikseli, których jasność jest powyżej lub poniżej pewnego progu. Biblioteka pobiera dowolny obraz jako dane wejściowe i zwraca tablicę obiektów Blob, z których każdy może powiedzieć o jego punktach krawędziowych i obwiedni.

16.9 Sandbox

Do tej pory każdy szkic, który stworzyliśmy w tej książce, mógł zostać opublikowany online. Być może masz już stronę internetową pełną pracy nad przetwarzaniem. Kiedy jednak zaczniemy pracować z kamerą wideo, napotkamy problem. Istnieją pewne ograniczenia bezpieczeństwa związane z apletami internetowymi i to jest coś, co zobaczymy tu i tam przez resztę książki. Aplet internetowy, na przykład, nie może połączyć się z kamerą wideo na komputerze użytkownika. W większości aplety nie mogą łączyć się z żadnymi urządzeniami lokalnymi. Ma sens, że istnieją wymogi bezpieczeństwa. Gdyby nie było, programista mógłby utworzyć aplet, który łączy się z twoim dyskiem twardym i usunie wszystkie twoje pliki, wyśle e-mail do wszystkich swoich przyjaciół i powie: „Sprawdź ten naprawdę fajny link! „Aplikacje nie mają wymagań bezpieczeństwa. W końcu możesz pobierać aplikacje, które kasują i formatują dyski twarde. Jednak pobieranie i instalowanie aplikacji jest inne niż po prostu pojawienie się adresu URL i załadowanie apletu. Jest to założony poziom zaufania do aplikacji. Nawiasem mówiąc, czy funkcja przetwarzania będzie działać w aplecie internetowym, jest wymieniona na każdej pojedynczej stronie odniesienia w sekcji „Użycie. ”Jeśli mówi„ Web ”, można go użyć w aplecie internetowym.

Projekt Lekcji Siódmej

Opracuj oprogramowanie lustrzane zawierające techniki widzenia komputerowego. Wykonaj następujące kroki.

Krok 1. Zaprojektuj wzór bez koloru. Może to być wzór statyczny (taki jak mozaika) lub ruchomy (taki jak przykład „bazgroły”) lub kombinacja.

Krok 2. Pokoloruj wzór zgodnie z obrazem JPG.

Krok 3. Zastąp JPG obrazami z kamery na żywo (lub nagranej szybki czas).

Krok 4. Korzystając z technik widzenia komputerowego, zmień zachowanie elementów wzorców zgodnie z właściwościami obrazu. Na przykład, być może jaśniejsze piksele powodują wirowanie kształtów lub piksele zmieniające wiele powodują, że kształty odlatują z ekranu i tak dalej.

Użyj miejsca podanego poniżej, aby naszkicować projekty, notatki i pseudokod dla twojego projektu.

Tekst

17.1 Skąd pochodzą String?

W części 15 zbadaliśmy nowy typ danych obiektowych wbudowany w środowisko przetwarzania do obsługi obrazów - PImage. W tej części przedstawimy kolejny nowy typ danych, inną klasę, którą otrzymamy za darmo z Przetwarzanie, o nazwie String. Klasa String nie jest zupełnie nową koncepcją. Mieliśmy do czynienia z ciągami znaków za każdym razem, gdy drukowaliśmy jakiś tekst w oknie wiadomości lub ładowaliśmy obraz z pliku. println („drukowanie jakiegoś tekstu do okna wiadomości!"); // Drukowanie ciągu znaków PImage img = loadImage ("filename.jpg"); // Używanie ciągu znaków dla nazwy pliku. Mimo to, mimo że użyliśmy String tu i tam, musimy jeszcze je w pełni zbadać i uwolnić ich potencjał. Aby zrozumieć pochodzenie ciągu, przypomnijmy sobie, skąd klasy pochodzą. Wiemy, że możemy tworzyć własne klasy (Zoog, Car itp.). Możemy użyć klas wbudowanych w środowisko przetwarzania, takich jak PImage. I wreszcie, w ostatniej części dowiedzieliśmy się, że możemy zaimportować dodatkowe biblioteki przetwarzania, aby użyć pewnych klas, takich jak Przechwytywanie lub Film. Niemniej jednak te klasy pochodzą z naszego życia w bańce przetwarzania. Musimy jeszcze wyjść w nieznane, świat tysięcy dostępnych klas Java. Zanim przejdziemy przez klif Java API (co zrobimy w części 23), warto zajrzeć przez krawędź i zbadać jeden najbardziej podstawowych i podstawowych klas Java, klasa String, której będziemy używać do przechowywania i manipulowania tekstem. Gdzie znajdziemy dokumentację dla klasy String? Aby poznać szczegóły dotyczące wbudowanych zmiennych, funkcji i klas, referencja Processing zawsze była naszym przewodnikiem. Chociaż technicznie jest to klasa Java, ponieważ klasa String jest tak powszechnie używana, przetwarzanie zawiera w swoim dokumencie dokumentację. Ponadto nie jest wymagane żadne oświadczenie o przywozie. http://www.processing.org/reference/String.html.

Ta strona zawiera tylko niektóre z dostępnych metod klasy String. Pełną dokumentację można znaleźć w witrynie Java firmy Sun: http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html

(Zwróć również uwagę na adres URL całego API Java: http://java.sun.com/j2se/1.4.2/docs/api). Nie omówimy jeszcze, jak korzystać z dokumentacji Java (zobacz Rozdział 23 po więcej), ale możesz zaostrzyć apetyt, odwiedzając i przeglądając powyższe linki.

17.2 Co to jest string?

String, w istocie, jest po prostu wymyślnym sposobem przechowywania tablicy znaków - gdybyśmy nie mieli klasy String, prawdopodobnie musielibyśmy napisać taki kod:

char [] sometext = {'H', 'e', 'l', 'l', 'o', '', 'W', 'o', 'r', 'l', 'd'};

Najwyraźniej byłby to królewski ból w procesie przetwarzania. Znacznie łatwiej jest wykonać następujące czynności i utworzyć obiekt String:

String sometext = " How do I make a String? Type characters between quotes! " ;

Z powyższego wynika, że string jest niczym więcej niż listą znaków w cudzysłowie. Niemniej jednak, są to tylko dane ciągu. Musimy pamiętać, że String jest obiektem z metodami (które można znaleźć na stronie odniesienia). Jest tak samo, jak obiekt PImage przechowuje dane powiązane z obrazem, a także ma funkcjonalność w postaci metod: copy (), loadPixels () i tak dalej. Przyjrzymy się bliżej metodom String w części 18. Oto kilka przykładów. Metoda charAt () zwraca indywidualny znak w łańcuchu o podanym indeksie. Ciągi są jak tablice, w których pierwszym znakiem jest indeks # 0!

Ćwiczenie 17-1: Jaki jest wynik następującego kodu?

String message = " a bunch of text here. " ;

char c = message.charAt(3);

println(c);

nną przydatną metodą jest długość (). Łatwo jest to pomylić z właściwością length tablicy. Jednak gdy pytamy o długość obiektu String, musimy użyć nawiasów, ponieważ wywołujemy funkcję o nazwie length (), a nie dostęp do właściwości o nazwie length.

String message = " This String is 34 characters long. " ;

println(message.length());

Exercise 17-2: Loop through and print every character of a String one at a time

String message = " a bunch of text here. ";

for (int i = 0; i < _____); i + +) {

char c = _____ ;

println(c);

}

Możemy również zmienić ciąg na wszystkie wielkie (lub małe litery) za pomocą metody toUpperCase ().

String uppercase = message.toUpperCase() ;

println(uppercase);

Możesz tu zauważyć coś dziwnego. Dlaczego po prostu nie powiedzieliśmy „message.toUpperCase ()", a następnie wydrukowaliśmy zmienną „message"? Zamiast tego przypisaliśmy wynik „message.toUpperCase ()" do nowej zmiennej o innej nazwie - „wielkie litery". Dzieje się tak, ponieważ ciąg jest szczególnym rodzajem obiektu. Jest niezmienny. Obiekt niezmienny to taki, którego danych nie można zmienić. Po utworzeniu ciągu pozostaje on taki sam przez całe życie. Za każdym razem, gdy chcemy zmienić String, musimy stworzyć nowy. Tak więc w przypadku konwersji na wielkie litery metoda toUpperCase () zwraca kopię obiektu String ze wszystkimi literami. Ostatnia metoda, którą omówimy w tym rozdziale, jest równa (). Możesz najpierw pomyśleć o porównaniu ciągów za pomocą operatora „= =".

String one = " hello";

String two = " hello";

println(one = = two);

Mówiąc technicznie, gdy „ = = „ jest używany z obiektami, porównuje adresy pamięci dla każdego obiektu. Nawet jeśli każdy ciąg zawiera te same dane „hello", jeśli są to różne instancje obiektów, wówczas „= =" może spowodować fałszywe porównanie. Funkcja equals () zapewnia, że sprawdzamy, czy dwa obiekty String zawierają dokładnie taką samą sekwencję znaków, niezależnie od tego, gdzie dane są przechowywane w pamięci komputera.

String one = "hello";

String two = "hello";

println (one.equals (two));

Chociaż obie powyższe metody zwracają poprawny wynik, bezpieczniej jest używać równych (). W zależności od tego, jak obiekty String są tworzone w szkicu, „ == „ nie zawsze działa.

Ćwiczenie 17-3: Znajdź duplikaty w następującej tablicy ciągów.

String words = { "I","love " , " coffee " , " I " , " love " , " tea " } ;

for (int i = 0; i < _____; i + +) {

for (int j = _; j < _____; j + +) {

if (_____) {

println(_____ + " is a duplicate. ");

}

}

}

Inną cechą obiektów String jest konkatenacja, łączenie dwóch łańcuchów razem. Łańcuchy są łączone za pomocą operatora „+”. Operator plus (+), oczywiście, zazwyczaj oznacza dodanie w przypadku liczb. W połączeniu z ciągami oznacza dołączenie.

String helloworld = " Hello " + " World " ;

Zmienne można również wprowadzać do łańcucha za pomocą konkatenacji.

int x = 10;

String message = "Wartość x to:" + x;

Widzieliśmy dobry przykład tego w części 15 podczas ładowania tablicy obrazów z numerowanymi nazwami plików.

Ćwiczenie 17-4: Połącz ciąg z podanych zmiennych, które wyprowadzają następujący komunikat.

Ten prostokąt ma szerokość 10 pikseli, wysokość 12 pikseli i siedzi dokładnie na (100 100).

float w = 10;

float h = 12;

float x = 100;

float y = 100;

String message = _____;

println(message);

17.3 Wyświetlanie tekstu

Będziemy nadal badać funkcje dostępne w ramach klasy String w rozdziale 18, który koncentruje się na analizowaniu i manipulowaniu ciągami znaków. To, czego do tej pory dowiedzieliśmy się o Strings,

jest wystarczające, aby skupić się na tym rozdziale: renderowanie tekstu. Najłatwiejszym sposobem wyświetlenia ciągu jest wydrukowanie go w oknie wiadomości. Jest to prawdopodobnie coś, co robiłeś tu i tam podczas debugowania. Na przykład, jeśli musisz znać położenie myszy poziomej, możesz napisać:

println (mouseX);

Lub jeśli musisz określić, że pewna część kodu została wykonana, możesz wydrukować opisową wiadomość.

println („Dostaliśmy się tutaj i drukujemy lokalizację myszy !!!”);

Jest to cenne narzędzie do debugowania, ale nie pomoże nam w wyświetlaniu tekstu dla użytkownika. Aby umieścić tekst na ekranie, musimy wykonać szereg prostych kroków.

1. Wybierz czcionkę, wybierając „Tools” → „Create Font. „To utworzy i umieści plik czcionki w katalogu danych. Zanotuj nazwę czcionki dla kroku 3. Przetwarzanie wykorzystuje specjalny format czcionki „vlw”, który używa obrazów do wyświetlania każdej litery. Z tego powodu powinieneś utworzyć czcionkę o rozmiarze, który chcesz wyświetlić.

2. Zadeklaruj obiekt typu PFont.

PFont f;

3. Załaduj czcionkę, odwołując się do nazwy czcionki w funkcji loadFont (). To powinno być zrobione tylko raz, zwykle w setup (). Podobnie jak w przypadku ładowania obrazu, proces ładowania czcionki do pamięci jest powolny i poważnie wpłynie na wydajność szkicu, jeśli zostanie umieszczony wewnątrz drawa ().

f = loadFont („ArialMT-16.vlw”);

4. Określ czcionkę za pomocą textFont (). textFont () pobiera jeden lub dwa argumenty, zmienną czcionki i rozmiar czcionki, która jest opcjonalna. Jeśli nie podasz rozmiaru czcionki, czcionka zostanie wyświetlona w oryginalnie załadowanym rozmiarze. Określenie rozmiaru czcionki różniącego się od rozmiaru załadowanej czcionki może skutkować tekstem w pikselach lub niskiej jakości.

textFont (f, 36);

5. Określ kolor za pomocą fill().

fill(0);

6. Wywołaj funkcję text (), aby wyświetlić tekst. (Ta funkcja jest podobna do rysunku lub rysunku, wymaga trzech argumentów - tekstu do wyświetlenia i współrzędnej x i y, aby wyświetlić ten tekst).

tekst („Mmmmm ... Strings ...”, 10,100);

Wszystkie kroki razem pokazano w przykładzie 17-1.

Przykład 17-1: Prosty tekst wyświetlany

```
PFont f; // STEP 2 Declare PFont variable

void setup() {

size(200,200);
```

```
f = loadFont( " ArialMT-16.vlw " ); // STEP 3 Load Font

}

void draw() {

background(255);

textFont(f,16); // STEP 4 Specify font to be used

fill(0); // STEP 5 Specify font color

text ( " Mmmmm... Strings ... " ,10,100); // STEP 6 Display Text

}
```

Czcionki można również tworzyć za pomocą funkcji createFont ().

```
myFont = createFont („Georgia”, 24, true);      //Argumenty funkcji createFont () to nazwa czcionki,
                                                //rozmiar czcionki i wartość logiczna, która umożliwia
                                                //wygładzanie (wygładzanie)
```

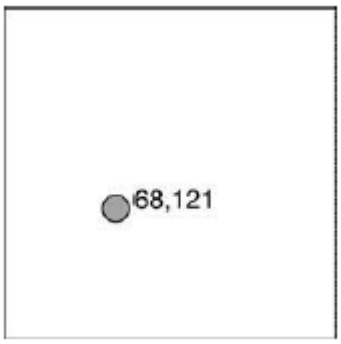
createFont () pozwala na użycie czcionki, która może być zainstalowana na lokalnym komputerze, ale nie jest dostępna jako część opcji czcionek Przetwarzanie. Ponadto createFont () umożliwia skalowanie czcionki do dowolnego rozmiaru bez patrzenia na piksele lub smugi. Aby uzyskać więcej informacji o createFont (), odwiedź odwołanie do przetwarzania:

http://processing.org/reference/createFont_.html. Możesz zobaczyć wszystkie dostępne czcionki za pomocą „PFont.list (). ”

```
println (PFont.list ()) //Wydrukuj wszystkie czcionki dostępne dla createFont () do okna wiadomości.
```

Dla celów demonstracyjnych createFont () będzie używany dla wszystkich przykładów, a nie dla loadFont ().

Ćwiczenie 17-5: Weź przykład odbijającej się piłki z części 5. Wyświetl współrzędne X i Y jako tekst obok piłki



17.4 Animacja tekstu

Teraz, gdy rozumiemy kroki wymagane do wyświetlenia tekstu, możemy zastosować inne koncepcje z tej książki, aby animować tekst w czasie rzeczywistym. Aby zacząć, spójrzmy na dwie bardziej przydatne funkcje przetwarzania związane z wyświetlaniem tekstu:

textAlign( ) —określa wyrównanie PRAWO, LEWO lub CENTRUM tekstu.

Przykład 17-2: Wyrównywanie tekstu

```
PFont f;

void setup() {

size(400,200);

f = createFont( "Arial",16,true);

}

void draw() {

background(255);

stroke(175);

line(width/2,0,width/2,height);

textFont(f);

fill(0);

textAlign(CENTER);

text( " This text is centered. " ,width/2,60);

textAlign (LEFT) ;        //textAlign () ustawia wyrównanie do wyświetlania tekstu. Wymaga jednego
                          //argumentu: CENTER, LEFT lub RIGHT.

text( " This text is left aligned. " ,width/2,100);

textAlign(RIGHT);

text( " This text is right aligned. " ,width/2,140);

}
```

textWidth () - Oblicza i zwraca szerokość dowolnego znaku lub ciągu tekstowego.

Powiedzmy, że chcemy utworzyć pasek wiadomości, w którym tekst przewija się na dole ekranu od lewej do prawej. Gdy nagłówek wiadomości opuszcza okno, pojawia się ponownie po prawej stronie i przewija ponownie. Jeśli znamy lokalizację x początku tekstu i znamy szerokość tego tekstu, możemy określić, kiedy nie jest on już widoczny. textWidth () daje nam tę szerokość. Na początek deklarujemy nagłówek, czcionkę i zmienne lokalizacji x, inicjując je w setup ().

```
// A headline

String headline = " New study shows computer programming lowers cholesterol. " ;

PFont f; // Global font variable

float x; // Horizontal location of headline

void setup() {

f = createFont( " Arial " ,16,true); // Loading font

x = width; // Initializing headline off-screen to the right
```

}

Funkcja draw () jest podobna do naszego przykładu odbijającej piłki w części 5. Najpierw wyświetlamy tekst w odpowiednim miejscu.

// Display headline at x location

textFont(f,16);

textAlign(LEFT);

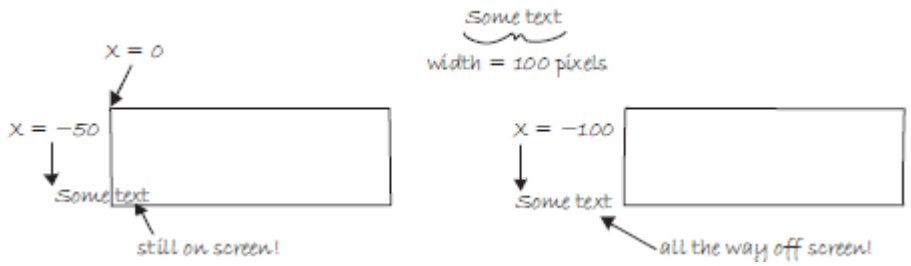
text(headline,x,180);

Zmieniamy x o wartość prędkości (w tym przypadku liczbę ujemną, aby tekst przesuwał się w lewo).

// Decrement x

x = x – 3;

Teraz pojawia się trudniejsza część. Łatwo było sprawdzić, kiedy okrąg dotarł do lewej strony ekranu. Po prostu pytamy: czy x jest mniejsze niż 0? Jednak z tekstem, ponieważ jest wyrównany do lewej, gdy x równa się zero, jest nadal widoczny na ekranie. Zamiast tego tekst będzie niewidoczny, gdy x jest mniejsze niż 0 minus szerokość tekstu. W takim przypadku resetujemy x z powrotem do prawej strony okna, to znaczy szerokość.



float w = textWidth(headline) ;

if (x < -w) { //Jeśli x jest mniejsze niż ujemna szerokość, to jest całkowicie wyłączone z ekranu

x = width;

}

Przykład 17-3 jest pełnym przykładem, który wyświetla inny nagłówek za każdym razem, gdy poprzedni nagłówek opuszcza ekran. Nagłówki są przechowywane w tablicy String.

Przykład 17-3: Przewijanie nagłówków

// An array of news headlines

String[] headlines = {

" Processing downloads break downloading record. ", //Wiele łańcuchów jest przechowywanych w
//tablicy.

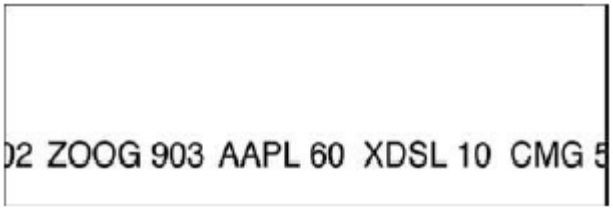
" New study shows computer programming lowers cholesterol. ",

} ;

```
PFont f; // Global font variable

float x; // Horizontal location

int index = 0;

void setup() {

size(400,200);

f = createFont( "Arial",16,true);

// Initialize headline offscreen

x = width;

}

void draw() {

background(255);

fill (0);

// Display headline at x location        //Określony ciąg z tablicy jest wyświetlany zgodnie z wartością
                                          //zmiennej „indeks”.

textFont(f,16);

textAlign (LEFT );

text(headlines[index],x,180);

// Decrement x

x = x – 3;

// If x is less than the negative width,

// then it is off the screen

float w = textWidth(headlines[index]);  //textWidth () służy do obliczania szerokości bieżącego ciągu.

if (x < -w) {

x = width;

index = (index + 1) % headlines.length;  //„Index” jest zwiększany, gdy bieżący łańcuch opuści ekran,
                                         //aby wyświetlić nowy ciąg.

}

}
```

Oprócz textAlign () i textWidth (), Processing oferuje także funkcje textLeading (), textMode () i textSize () dla dodatkowych funkcji wyświetlania. Te funkcje nie są konieczne w przykładach opisanych w tym rozdziale, ale możesz je eksplorować samodzielnie w odnośniku Przetwarzanie.

Ćwiczenie 17-6: Stwórz kurs giełdowy, który ciągle się powtarza. Gdy ostatni zapis wchodzi do okna, pierwszy zapis pojawia się natychmiast po jego prawej stronie.



17.5 Mozaika tekstu

Łącząc to, czego dowiedzieliśmy się z rozdziałów 15 i 16 na temat tablicy pikseli, możemy użyć pikseli obrazu do stworzenia mozaiki znaków. Jest to rozszerzenie kodu lustrzanego wideo w części 16.

Przykład 17-4: Lustro tekstu

```
import processing.video.*;

// Size of each cell in the grid, ratio of window size to video size

int videoScale = 14;

// Number of columns and rows in our system

int cols, rows;

// Variable to hold onto capture object

capture video;

// A String and Font

String chars = " helloworld" ;     //Tekst źródłowy użyty w mozaice. Może być dłuższy ciąg tworzyć
                                   //bardziej interesujące wyniki.
```

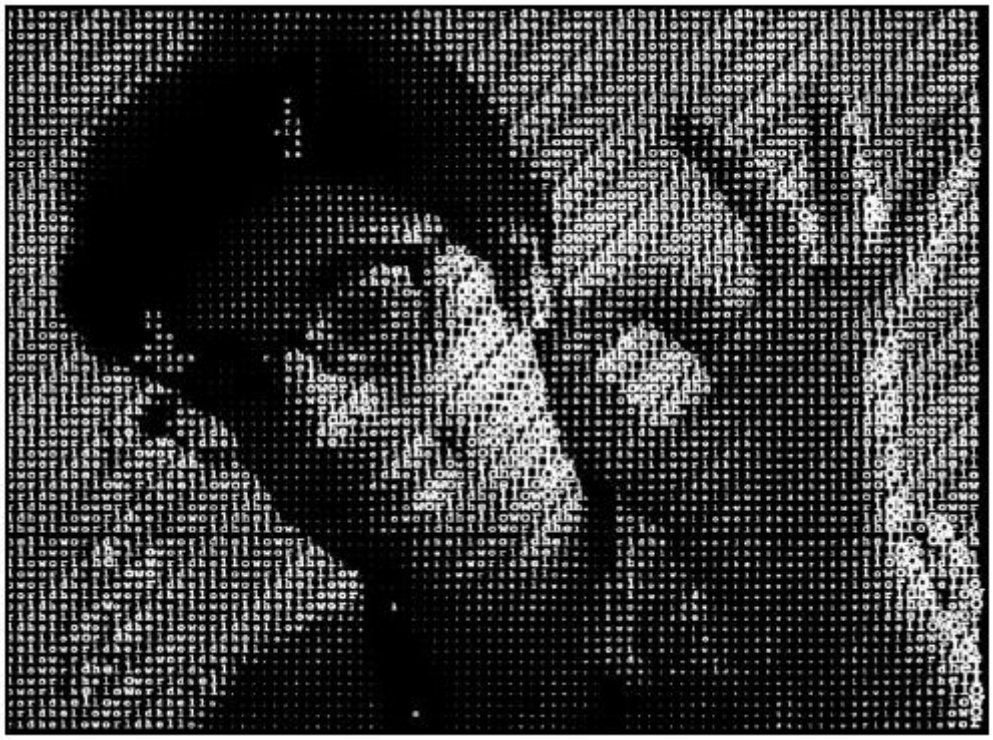
```
PFont f;

void setup() {

size(640,480);

//set up columns and rows

cols = width/videoScale;

rows = height/videoScale;

video = new Capture(this,cols,rows,15);

// Load the font          //Używanie czcionki „o stałej szerokości”. W większości czcionek poszczególne
                          //znaki mają różne szerokości. W czcionce o stałej szerokości wszystkie znaki
                          //mają tę samą szerokość. Jest to przydatne, ponieważ zamierzamy wyświetlić
                          //jedną literę w równych odstępach czasu.

f = loadFont (" Courier-Bold-20.vlw ");

}

void draw() {

background(0);

// Read image from the camera

if (video.available()) {

video.read();

}

video.loadPixels();

// Use a variable to count through chars in String

int charcount = 0;

// Begin loop for rows

for (int j = 0; j < rows; j + + ) {

// Begin loop for columns

for (int i = 0; i < cols; i + + ) {

// Where are we, pixel-wise?

int x = i*videoScale;

int y = j*videoScale;

color c = video.pixels[i + j*video.width];

// Instead of a rectangle

textFont(f);
```

fill(c); //Jeden znak z tekstu źródłowego jest wyświetlany w kolorze odpowiednio do położenia //piksela. Zmienna licznikowa - „charcount” - służy do przechodzenia przez ciąg znaków po //Ćwiczenie 17-7: Utwórz mozaikę tekstu wideo, w której każda litera ma kolor biały, ale rozmiar każdej litery zależy od jasności pikseli. Im jaśniejszy piksel, tym większy. Oto trochę kodu z wnętrza pętli pikseli (z kilkoma pustymi miejscami), abyś mógł zacząć.jednym znaku naraz.

```
text(chars.charAt(charcount),x,y);

// Go on to the next character

charcount = (charcount + 1) % chars.length();

}

}

}
```

Ćwiczenie 17-7: Utwórz mozaikę tekstu wideo, w której każda litera ma kolor biały, ale rozmiar każdej litery zależy od jasności pikseli. Im jaśniejszy piksel, tym większy. Oto trochę kodu z wnętrza pętli pikseli (z kilkoma pustymi miejscami), abyś mógł zacząć.

```
float b = brightness(video.pixels[i + j*video.width]);

float fontSize = ____ * (_____ / _____);

textSize(fontSize);
```



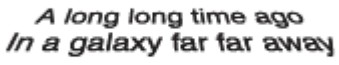
17.6 Obracanie tekstu

Tłumaczenie i obrót (jak pokazano w rozdziale 14) można również zastosować do tekstu. Na przykład, aby obrócić tekst wokół jego środka, przetłumacz na punkt początkowy i użyj textAlign (CENTER) przed wyświetleniem tekstu.

Przykład 17-5: Obracanie tekstu

```
PFont f;

String message = " this text is spinning " ;

float theta;

void setup() {

size(200,200);

f = createFont( " Arial " ,20,true);

}

void draw() {

background(255);

fill (0);

textFont(f); // Set the font

translate(width/2,height/2); // Translate to the center

rotate(theta); // Rotate by theta

textAlign( CENTER) ;

text(message,0,0);        //Tekst jest wyrównany do środka i wyświetlany w (0,0) po translacji i obrocie.

theta + = 0.05; // Increase rotation

}
```

Ćwiczenie 17-8: Wyświetl tekst wyśrodkowany i obrócony, aby wyglądał płasko. Przewiń tekst w dal.

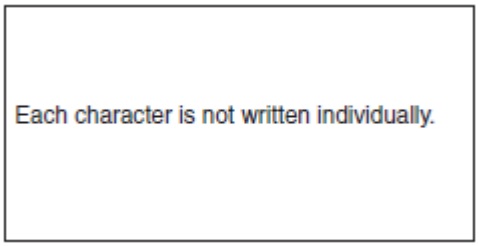



```
String info = " A long long time ago\nIn a galaxy far far away ";

PFont f;          //„\" Oznacza „nową linię". W Javie niewidzialne znaki mogą być włączone do łańcucha
                  //za pomocą „Sekwencja ucieczki" – wstecz ukośnik „/" i znak. Oto jeszcze kilka.

                  // \n - nowa linia

                  // \r - powrót karetki

                  // \t - tabulator
```

```
                //\'- pojedynczy cytat
                //\"-cudzysłów
                // \\- ukośnik do tyłu
float y = 0;
void setup() {
size(400,200,P3D);
f = createFont( "Arial",20*4,true);
}
void draw() {
background(255);
fill(0);
translate(_____,_____);
_____ (_____);
textFont(f);
textAlign(CENTER);
text(info, _____,_____);
y--;
}
```

17.7 Wyświetl znak tekstu po znaku.

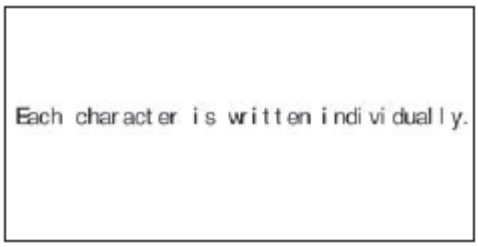
W niektórych aplikacjach graficznych wymagane jest wyświetlanie tekstu z każdym znakiem osobno. Na przykład, jeśli każda postać musi się poruszać niezależnie, to po prostu powiedzenie tekstu („pęczek liter", 0,0) nie zrobi. Rozwiązaniem jest przejście przez łańcuch, wyświetlając po jednym znaku po kolei. Zacznijmy od przykładu, który wyświetla tekst naraz. Patrz Rysunek




```
PFont f;
String message = " Each character is not
written individually." ;
void setup() {
```

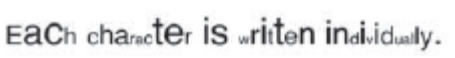
```
size(400,200);
f = createFont( " Arial " ,20,true);
}
void draw() {
background(255);
fill(0);
textFont(f);
text(message,10,height/2);// Wyświetlanie bloku tekstu jednocześnie za pomocą text()
}
```

Przepisujemy kod, aby wyświetlić każdy znak w pętli, używając funkcji charAt (). Patrz rysunek




```
int x = 10;          //Pierwszy znak znajduje się w pikselu 10
for (int i = 0; i < message.length(); i + + ) {
text(message.charAt(i) ,x, height/2);
x + = 10;          //Każdy znak jest wyświetlany pojedynczo za pomocą funkcji charAt ().
}          //Wszystkie znaki są oddalone od siebie o 10 pikseli.
```

Wywołanie funkcji text () dla każdego znaku pozwoli nam na większą elastyczność w przyszłych przykładach (do kolorowania, rozmiaru i umieszczania znaków w pojedynczym ciągu). Ten przykład ma ładną główną wadę. Tutaj lokacja x jest zwiększana o 10 pikseli dla każdej postaci. Chociaż jest to w przybliżeniu poprawne, ponieważ każdy znak nie ma dokładnie 10 pikseli szerokości, odstępy są wyłączone. Prawidłowe odstępy można uzyskać za pomocą funkcji textWidth (), jak pokazano w poniższym kodzie. Zauważ, że ten przykład osiąga właściwe odstępy, nawet jeśli każdy znak jest losowy! Patrz rysunek



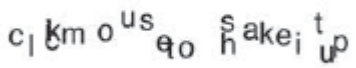

```
int x = 10;
for (int i = 0; i < message.length(); i + + ) {
textSize(random(12,36));
text(message.charAt(i) ,x, height/2);
x + = textWidth(message.charAt(i));
```

} //textWidth () poprawnie wypisuje znaki.

Ćwiczenie 17-9: Używanie textWidth (), ponów przykład 17-4 (tekst „mirror”), aby użyć czcionki o nieokreślonej szerokości z odpowiednim odstępem między znakami. Poniższy obraz używa Arial.

Jest to metodologia „litera po literze”, którą można również zastosować do szkicu, w którym postacie ze sznurka poruszają się niezależnie od siebie. Poniższy przykład używa zorientowanego obiektowo projektu, aby każdy znak z oryginalnego Ciągu był obiektem Listu, co umożliwia zarówno jego wyświetlanie we właściwym miejscu, jak i poruszanie się po ekranie indywidualnie.

Przykład 17-6: Rozpad tekstu



```
PFont f;

String message = " click mouse to shake it up ";

// An array of Letter objects

Letter[] letters;

void setup() {

size(260,200);

// Load the font

f = createFont( "Arial",20,true);

textFont(f);
```

```
// Create the array the same size as the String
letters = new Letter[message.length()];
// Initialize Letters at the correct x location
int x = 16;
for (int i = 0; i < message.length(); i + + ) {
letters[i] = new Letter(x,100,message.charAt(i));        //Obiekty literowe są inicjowane ich
                                                         //położeniem w ciągu jak również jaki znak
                                                         //powinien wyświetlać.
x + = textwidth(message.charAt(i));
}
}
void draw() {
background(255);
for (int i = 0; i < letters.length; i + + ) {
// Display all letters
letters[i].display();
// If the mouse is pressed the letters shake
// If not, they return to their original location
if (mousePressed) {
letters[i].shake();
} else {
letters[i].home();
}
}
}
// A class to describe a single Letter
class Letter {
char letter;
// The object knows its original " home " location
float homex,homey;     //Obiekt wie o swojej pierwotnej „domowej” lokalizacji w Ciągu tekstu, a także
                       //o jego bieżących (x, y) lokalizacjach, które powinny poruszać się po ekranie.
// As well as its current location
```

```
float x,y;

Letter (float x_, float y_, char letter_) {

homex = x = x_;

homey = y = y_;

letter = letter_;

}

// Display the letter

void display() {

fill(0);

textAlign(LEFT);

text(letter,x,y);

}

// Move the letter randomly

void shake() {

x + = random(-2,2);

y + = random(-2,2);

}

// Return the letter home

void home() {    //W dowolnym momencie bieżąca lokalizacja może zostać przywrócona do lokalizacji
                 //domowej przez wywołanie funkcji home ().

x = homex;

y = homey;

}

}
```

Metoda znak po znaku pozwala również wyświetlać tekst wzdłuż krzywej. Zanim przejdziemy do liter, spójrzmy najpierw, jak narysujemy serię skrzynek wzdłuż krzywej. Ten przykład w znacznym stopniu wykorzystuje funkcje trygonometryczne opisane w części 13.

Przykład 17-7: Pola wzdłuż krzywe

```
PFont f;

// The radius of a circle

float r = 100;

// The width and height of the boxes
```

```
float w = 40;
float h = 40;
void setup() {
size(320,320);
smooth();
}
void draw() {
background(255);
// Start in the center and draw the circle
translate(width /2, height /2);
noFill();
stroke(0);
ellipse(0, 0, r*2, r*2);    //Nasza krzywa jest okręgiem o promieniu r pośrodku okna
// 10 boxes along the curve
int totalBoxes = 10;
// We must keep track of our position along the curve
float arclength = 0;
// For every box
for (int i = 0; i < totalBoxes; i + + ) {
// Each box is centered so we move half the width
arclength + = w/2;         //Poruszamy się wzdłuż krzywej w zależności od szerokości pudełka
// Angle in radians is the arclength divided by the radius
float theta = arclength /r;
pushMatrix();
// Polar to cartesian coordinate conversion
translate(r*cos(theta) , r*sin(theta));
// Rotate the box
rotate(theta);
// Display the box
fill(0 , 100);
rectMode(CENTER);
```

```
rect(0 , 0 , w,h);

popMatrix();

// Move halfway again

arclength + = w/2;

}

}
```

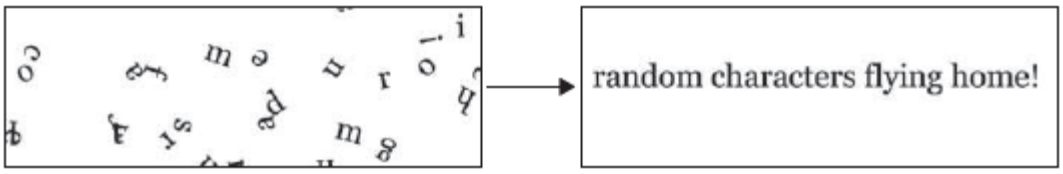
Nawet jeśli znajdziesz matematykę tego przykładu jako trudną, rysunek poniżej powinien ujawnić następny krok. To, co musimy zrobić, to zastąpić każdą skrzynkę postacią ze sznurka, który znajduje się wewnątrz pudełka. A ponieważ nie wszystkie znaki mają tę samą szerokość, zamiast używać zmiennej „w", która pozostaje stała, każde pole będzie miało zmienną szerokość wzdłuż krzywej zgodnie z funkcją textWidth ()

Przykład 17-8: Znaki wzdłuż krzywej

```
// The message to be displayed

String message = " text along a curve " ;

PFont f;

// The radius of a circle

float r = 100;

void setup() {

size(320,320);

f = createFont( " Georgia ",40,true);

textFont(f);

textAlign(CENTER);        //Tekst musi być wyśrodkowany!

smooth();

}

void draw() {

background(255);

// Start in the center and draw the circle

translate(width / 2, height / 2);
```

```
noFill();
stroke(0);
ellipse(0, 0, r*2, r*2);
// We must keep track of our position along the curve
float arclength = 0;
// For every box
for (int i = 0; i < message.length(); i + + ) {
// The character and its width    //Zamiast stałej szerokości sprawdzamy szerokość każdego znaku.
char currentChar = message.charAt(i);
float w = textWidth(currentChar);
// Each box is centered so we move half the width
arclength + = w/2;
// Angle in radians is the arclength divided by the radius
// Starting on the left side of the circle by adding PI
float theta = PI + arclength / r;
pushMatrix();
// Polar to cartesian coordinate conversion
translate(r*cos(theta), r*sin(theta));     //Konwersja biegunowa na kartezjańska pozwala nam znaleźć
                                           //punkt krzywej
// Rotate the box
rotate(theta + PI/2); // rotation is offset by 90 degrees
// Display the character
fill(0);
text(currentChar,0,0);
popMatrix();
// Move halfway again
arclength + = w/2;
}
}
```

Ćwiczenie 17-10: Utwórz szkic, który zaczyna się od znaków losowo rozrzuconych (i obróconych). Niech powoli wrócą do swojej „domowej” lokalizacji. Użyj podejścia obiektowego, jak widać w przykładzie 17-6.



Jednym ze sposobów rozwiązania tego problemu jest użycie interpolacji. Interpolacja odnosi się do procesu obliczania wartości pomiędzy dwoma danymi danych. W tym ćwiczeniu chcemy poznać pośrednie lokalizacje x (i y lokalizacje) od losowego punktu początkowego do docelowej lokalizacji w ciągu. Jedną z metod interpolacji jest po prostu przyjęcie średniej. Atrament tego, jak idzie do ściany, zawsze robiąc krok do połowy.

x = (x + targetX) / 2;

y = (y + targetY) / 2;

Inną możliwością jest po prostu przejście 10% drogi.

x = 0.9 * x + 0.l * targetX;

y = 0,9 * y + 0,1 * targetY;

Funkcja lerp () przetwarzania wykona dla Ciebie interpolację. Odwiedź stronę referencyjną, aby uzyskać więcej informacji: http://processing.org/reference/lerp_.html.

Zastanów się nad interaktywnym szkicem. Czy możesz przesuwać litery za pomocą myszy?

Wprowadzanie danych

18.1 Manipulowanie łańcuchami

W części 17 poruszyliśmy kilka podstawowych funkcji dostępnych w klasie Java String, takich jak charAt (), toUpperCase (), equals () i length (). Te funkcje są udokumentowane w Przetwarzaniu

strona referencyjna dla ciągów. Niemniej jednak, aby wykonać bardziej zaawansowane techniki analizowania danych, będziemy musieli zbadać kilka dodatkowych funkcji manipulacji ciągiem dokumentów udokumentowanych w interfejsie API Java:

http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html

Przyjrzyjmy się bliżej następującym dwóm funkcjom String: indexOf () i substring (). indexOf () lokalizuje sekwencję znaków w ciągu. Przyjmuje jeden argument, ciąg wyszukiwania i zwraca wartość liczbową, która odpowiada pierwszemu wystąpieniu ciągu wyszukiwania wewnątrz przeszukiwanego ciągu.

String search = " def";

String toBeSearched = " abcdefghi";

int index = toBeSearched.indexOf(search); <- Wartość indeksu w tym przykładzie wynosi 3

Łańcuchy są podobne do tablic, ponieważ pierwszym znakiem jest liczba zero, a ostatnim znakiem jest długość ciągu minus jeden.

Ćwiczenie 18-1: Przewiduj wynik poniższego kodu.

String sentence = " The quick brown fox jumps over the lazy dog. " ;

println(sentence.indexOf(" quick ")); _____

println(sentence.indexOf(" fo ")); _____

println(sentence.indexOf(" The ")); _____

println(sentence.indexOf(" blah blah ")); _____

Jeśli utkniesz w ostatnim wierszu ćwiczenia 18-1, dzieje się tak, ponieważ nie możesz w żaden sposób znać odpowiedzi bez sprawdzenia odniesienia do Javy (lub dokonania zgadywanego zgadywania). Jeśli nie można znaleźć ciągu wyszukiwania, indexOf () zwraca -1. Jest to dobry wybór, ponieważ „-1" nie jest uzasadnioną wartością indeksu w samym ciągu, a zatem może wskazywać „nie znaleziono. „Nie ma ujemnych indeksów w ciągu znaków lub tablicy. Po znalezieniu szukanego wyrażenia w ciągu możemy chcieć oddzielić część ciągu, zapisując go w innej zmiennej. Część łańcucha jest znana jako sstrstring, a podciągi są tworzone za pomocą funkcji substring (), która pobiera dwa argumenty, indeks początkowy i indeks końcowy. substring () zwraca podciąg między dwoma indeksami

String alphabet = " abcdefghi " ;

String sub = alphabet.substring(3,6); <- Podciąg String to teraz „def"

Zauważ, że podłańcuch zaczyna się od określonego indeksu początkowego (pierwszy argument) i rozciąga się na znak na końcu indeksu (drugi argument) minus jeden. Wiem. Wiem. Czy nie byłoby łatwiej po prostu przenieść podciąg z indeksu początkowego aż do indeksu końcowego? Chociaż może to początkowo wydawać się prawdą, w rzeczywistości całkiem wygodnie jest zatrzymać się przy indeksie końcowym minus jeden. Na przykład, jeśli kiedykolwiek chcesz utworzyć podciąg, który

rozciąga się na koniec łańcucha, możesz po prostu przejść całą drogę do thestring.length (). Ponadto, przy indeksie końcowym minus jeden oznaczającym koniec, długość podłańcucha jest łatwo obliczana jako indeks końcowy minus indeks początkowy.

Ćwiczenie 18-2: Wypełnij puste pola, aby uzyskać podciąg „fox jumps over the lazy dog” (bez kropki).

String sentence = " The quick brown fox jumps over the lazy dog. " ;

int foxIndex = sentence.indexOf(_____);

int periodIndex = sentence.indexOf(" . ");

String sub = _____ ·_____(_____,_____);

18.2 Dzielenie i łączenie

W części 17 zobaczyliśmy, jak można łączyć łańcuchy (określane jako „konkatenacja”) za pomocą operatora „+”. Przyjrzyjmy się przykładowi, który używa konkatenacji, aby uzyskać dane wejściowe użytkownika z klawiatury

```
PFont f;

// Variable to store text currently being typed

String typing = " " ;      //W przypadku wprowadzania z klawiatury używamy dwóch zmiennych. Jedna
                           //zapisze tekst w trakcie pisania. Druga zachowa kopię wpisanego tekstu po
                           //naciśnięciu klawisza Enter.

// Variable to store saved text when return is hit

String saved = " " ;

void setup() {

size(300,200);

f = createFont( "Arial",16,true);

}

void draw() {

background(255);

int indent = 25;

// Set the font and fill for text

textFont(f);

fill(0);

// Display everything

text("Click in this applet and type. \nHit return to save what you typed. ",indent,40);

text(typing,indent,90);

text(saved,indent,130);
```

```
}

void keyPressed() {

// If the return key is pressed, save the String and clear it

if (key = = '\n') {

saved = typing; // Łańcuch można wyczyścić, ustawiając go na „”.

typing = " " ;

// Otherwise, concatenate the String

} else {

typing = typing + key;    //Każdy znak wpisany przez użytkownika jest dodawany na końcu zmiennej
                          //String.

}

}
```

Ćwiczenie 18-3: Utwórz szkic, który rozmawia z użytkownikiem. Na przykład, jeśli użytkownik wprowadzi „koty”, szkic może odpowiedzieć „Jak czujesz się u kotów? ”

Procesing ma dwie dodatkowe funkcje, które ułatwiają łączenie ciągów znaków (lub odwrotnie, dzielenie ich). W szkicach, które obejmują parsowanie danych z pliku lub sieci, często otrzymujemy te dane w postaci tablicy ciągów lub jednego długiego ciągu. W zależności od tego, co chcemy osiągnąć, warto wiedzieć, jak przełączać się między tymi dwoma trybami przechowywania. To właśnie tam te dwie nowe funkcje, split() i join(), przyda się. „Jeden długi ciąg lub tablica ciągów” ← → {„jeden”, „długi”, „ciąg”, „lub”, „tablica”, „z”, „ciągi”)

Przyjrzyjmy się funkcji split (). split () oddziela dłuższy ciąg znaków na tablicę ciągów znaków, w oparciu o znak podziału znany jako separator. Wymaga dwóch argumentów: łańcucha do podziału i separatora. (Separator może być pojedynczym znakiem lub ciągiem.)

```
// Dzielenie ciągu na podstawie spacji

String spaceswords = " The quick brown fox jumps over the lazy dog. " ;

String[] list = split(spaceswords, " " );

for (int i = 0; i < list.length; i + + ) {

println(list[i] + " " + i);

}
```

Oto przykład użycia przecinka jako separatora.

```
// Dzielenie ciągu na podstawie przecinków

String commaswords = " The,quick,brown,fox,jumps,over,the,lazy,dog. " ;

String[] list = split(commaswords, ‘ , ’ );

for (int i = 0; i < list.length; i + + ) {
```

```
println(list[i] + " " + i);

}
```

Jeśli chcesz użyć więcej niż jednego separatora, aby podzielić tekst, musisz użyć funkcji Przetwarzanie splitTokens (). splitTokens () działa identycznie, jest split () z jednym wyjątkiem: dowolny znak, który pojawia się w łańcuchu jako ogranicznik.

```
// Dzielenie ciągu na podstawie wielu ograniczników

String stuff = " hats & apples, cars + phones % elephants dog. " ; // Kropka  jest ustawiana jako separator
                                                                   //i dlatego nie zostanie uwzględniony
                                                                   //w ostatnim ciągu w tablicy: „dog”.

String[] list = splitTokens(stuff, " & , + . " );

for (int i = 0; i < list.length; i + + ) {

println(list[i] + " " + i);

}
```

Ćwiczenie 18-4: Wypełnij powyższy kod w oknie wiadomości przetwarzania:

hats_____

_____

_____

_____

_____

Jeśli dzielimy liczby w łańcuchu, wynikowa tablica może zostać przekształcona w tablicę liczb całkowitych funkcją int() Processing.

```
// Oblicz sumę listy liczb w Sting

String numbers = " 8,67,5,309 " ;

// Konwertowanie tablicy String na tablicę int

int[] list = int(split(numbers, ' , ' ));      //Liczby w łańcuchu nie są liczbami i nie mogą być używane w
                                               //operacjach matematycznych, chyba że najpierw je
                                               //przekonwertujemy.

int sum = 0;

for (int i = 0; i < list.length; i + + ) {

sum = sum + list[i];

}

println(sum);
```

Odwrotność split() to join). join() pobiera tablicę łańcuchów i łączy je w jeden długi łańcuch. Funkcja join() pobiera również dwa argumenty: tablicę do połączenia i separator. Separator może być pojedynczym znakiem lub ciągiem znaków.

Rozważ następującą tablicę:

String [] lines = {„It", „was", „a", „dark", „and", „stormy", „night"). };

Używając operatora + wraz z pętlą for, możemy połączyć Ciąg w następujący sposób:

// Ręczna konkatenacja

String onelongstring = "";

for (int i = 0; i <lines.length; i + +) {

onelongstring = onelongstring + linie [i] + "";

}

Funkcja join() pozwala nam jednak ominąć ten proces, osiągając ten sam wynik tylko w jednym wierszu kodu.

// Korzystanie z join() Processing

String onelongstring = join (linie, „");

Ćwiczenie 18-5: Podziel następujący ciąg na tablicę liczb punktów flotacyjnych i oblicz średnią.

String floats = " 5023.23:52.3:10.4:5.9, 901.3---2.3 ";

float[] numbers = _____(_____(_____, "_____"));

float total = 0;

for (int i = 0; i < numbers.length; i + +) {

_____ += _____;

}

float avg = _____;

18.3 Odczytywanie i zapisywanie plików tekstowych

Dane mogą pochodzić z wielu różnych miejsc: stron internetowych, kanałów informacyjnych, baz danych i tak dalej. Podczas opracowywania aplikacji, która obejmuje źródło danych, takie jak wizualizacja danych, ważne jest, aby oddzielić logikę tego, co program ostatecznie zrobi z danymi pochodzącymi z pobierania samych danych. , szczególnie przydatne jest rozwijanie danych „fałszywych" lub „fałszywych". Zgodnie z naszą mantrą „krok po kroku", gdy mięso programu zostanie uzupełnione fałszywymi danymi, możesz skupić się wyłącznie na tym, jak pobrać rzeczywiste dane z prawdziwego źródła. Będziemy stosować ten model w tej sekcji, pracując z najprostszymi sposobami pobierania danych: odczytywaniem z pliku tekstowego. Pliki tekstowe mogą być używane jako bardzo prosta baza danych (możemy przechowywać ustawienia programu, listę najlepszych wyników, liczby dla wykresu itp.) Lub symulować bardziej złożone źródło danych. Aby utworzyć plik tekstowy, możesz użyć dowolnego prostego edytora tekstu. Robi to Notatnik Windows lub Mac OS X TextEdit, po prostu upewnij się, że formatujesz plik jako „zwykły tekst. "Wskazane jest również nazywanie plików

tekstowych rozszerzeniem„ .txt ”, aby uniknąć nieporozumień. Podobnie jak w przypadku plików graficznych w części 15, te pliki tekstowe powinny być umieszczone w katalogu „dane” szkicu, aby mogły zostać rozpoznane przez szkic przetwarzania. Po wprowadzeniu pliku tekstowego funkcja loadStrings () przetwarzania odczytuje zawartość pliku w tablicy String. Poszczególne linie tekstu w pliku stają się pojedynczymi elementami tablicy.

```
String[] lines = loadStrings(" file.txt " );

println( " there are " + lines.length + " lines "    //Ten kod wyświetla wszystkie linie z pliku tekstu
                                                      //źródłowego

for (int i = 0; i < lines.length; i + + ) {

println(lines[i]);

}
```

Aby uruchomić kod, utwórz plik tekstowy o nazwie „file.txt”, wpisz kilka wierszy w tym pliku i umieść go w katalogu danych szkicu.

Ćwiczenie 18-6: Przepisz przykład 17-3, aby załadował nagłówki z pliku tekstowego.

Tekst z pliku danych można wykorzystać do wygenerowania prostej wizualizacji.

```
int[] data;

void setup() {

size(200,200);

// Load text file as a String

String[] stuff = loadStrings( "data.txt");  //Tekst z pliku jest ładowany do tablicy. Ta tablica zawiera
                                            //jeden element, ponieważ plik ma tylko jedną linię. Ten
                                            //element jest następnie dzielony na tablicę int.

// Convert string into an array of integers

// using ',' as a delimiter

data = int(split(stuff[0], ','));

}

void draw() {

background(255);

stroke(0);

for (int i = 0; i < data.length; i + + ) {

fill(data[i]);       // Tablica int’ów służy do ustawiania koloru i wysokości każdego prostokąta

rect(i*20,0,20,data[i]);

}

noLoop();
```

}

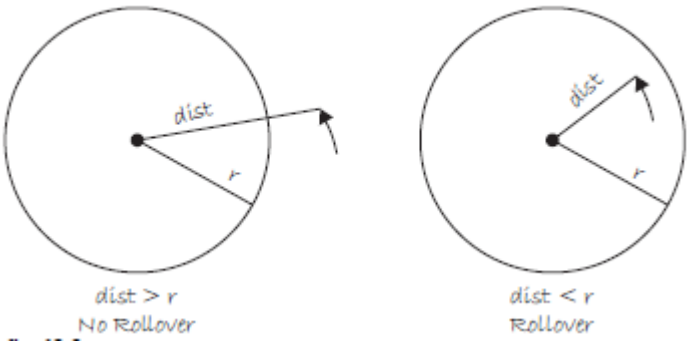
Możemy również użyć zawartości pliku tekstowego do inicjalizacji obiektów, przekazując dane z pliku tekstowego do konstruktora obiektów. W przykładzie 18-3 plik tekstowy ma 10 wierszy, przy czym każda linia reprezentuje jedno wystąpienie obiektu z wartościami dla zmiennych tego obiektu oddzielonych przecinkami

Przykład 18-3: Tworzenie obiektów z pliku tekstowego

```
Bubble[] bubbles;

void setup() {

size(200,200);

smooth();

// Load text file as an array of String

String[] data = loadStrings( "data.txt");

bubbles = new Bubble[data.length];     //Rozmiar tablicy obiektów Bubble jest określony przez
                                       //całkowitą liczbę linii w pliku tekstowym.

for (int i = 0; i < bubbles.length; i + + ) {

float[] values = float(split(data[i], ","));  //Każda linia jest podzielona na tablicę liczb
                                              //zmiennoprzecinkowych.

bubbles[i] = new Bubble(values[0],values[1],values[2]);

}                                      //Wartości w tablicy są przekazywane do konstruktora klasy Bubble.

}

void draw() {

background(100);

// Display and move all bubbles

for (int i = 0; i < bubbles.length; i + + ) {

bubbles[i].display();

bubbles[i].drift();

}

}

// A Class to describe a "Bubble"

class Bubble {

float x,y;

float diameter;

float speed;
```

```
float r,g;
// The constructor initializes color and size
// Location is filled randomly
Bubble(float r_, float g_, float diameter_) {
x = random(width);
y = height;
r = r_;
g = g_;
diameter = diameter_;
}
// Display the Bubble
void display() {
stroke(255);
fill(r,g,255,150);
ellipse(x,y,diameter,diameter);
}
// Move the bubble
void drift() {
y + = random(–3,–0.1);
x + = random(–1,1);
if (y < -diameter*2) {
y = height + diameter*2;
}
}
}
```

Teraz, gdy czujemy się dobrze z pomysłem załadowania informacji z pliku tekstowego w celu zainicjowania szkiców Processing, jesteśmy gotowi zadać następujące pytanie: Co, jeśli chcemy zapisać informacje, aby można je było załadować przy następnym uruchomieniu programu ? Załóżmy na przykład, że chcemy zmienić przykład 18-3, aby pęcherzyki zmieniały się po najechaniu myszą. (Pracowaliśmy nad najazdami z prostokątem wcześniej w ćwiczeniu 5-5 i przykładzie 9-11, ale ten przykład najazdu użyje okręgu.)



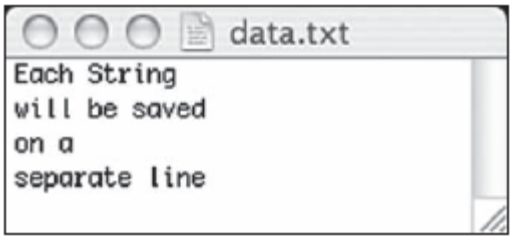

```
boolean rollover(int mx, int my) {          //Funkcja rollover () w klasie Bubble zwraca wartość logiczną
                                            //(true lub false) w zależności od tego, czy argumenty (mx, my)
                                            //znajdują się wewnątrz okręgu.

if (dist(mx,my,x,y) < diameter/2) {

return true;

} else {

return false;

}

}
```

Funkcja rollover() sprawdza, czy odległość między danym punktem (mx, my) a lokalizacją bąbelka (x, y) jest mniejsza niż promień okręgu; promień jest zdefiniowany jako średnica podzielona przez dwa. Jeśli to prawda, to ten punkt (mx, mój) musi znajdować się wewnątrz okręgu. Wywołanie tej funkcji przy użyciu lokalizacji myszy jako argumentów pozwala nam przetestować, czy mysz w rzeczywistości przetacza się nad którymkolwiek z bąbelków.

```
for (int i = 0; i < bubbles.length; i + + ) {

bubbles[i].display();

bubbles[i].drift();

if (bubbles[i].rollover(mouseX,mouseY)) {

bubbles[i].change();

}

}
```

Po zaimplementowaniu funkcji change () w celu dostosowania zmiennych Bańki możemy zapisać nowe informacje w pliku tekstowym za pomocą funkcji saveStrings () Przetwarzania. saveStrings () jest zasadniczo przeciwieństwem loadStrings (), odbierającym nazwę pliku fi i tablicę String oraz zapisującą tę tablicę do pliku

```
String[] stuff = { " Each String ", "will be saved ", "on a ", "separate line " } ;

saveStrings( "data.txt", stuff);    //Ten kod utworzy plik tekstowy na rysunku
```



saveStrings () nie przechowuje jednak pliku tekstowego w folderze danych, ale umieszcza go w lokalnym katalogu szkiców. Jeśli chcemy, aby plik został zapisany w katalogu danych, musimy określić ścieżkę. Ponadto, jeśli plik już istnieje, zostanie zastąpiony. Wiedząc o tym, możemy stworzyć funkcję saveData () dla szkicu Bubbles, który przepisuje „data.txt" z właściwościami obiektów w ich bieżącym stanie. W tym przykładzie zapiszemy nowe dane po każdym naciśnięciu myszy.

```
void saveData() {

String[] data = new String[bubbles.length];     // Najpierw tworzymy tablicę ciągów o rozmiarze
                                                 równym całkowitej liczbie obiektów Bubble.

for (int i = 0; i < bubbles.length; i + + ) {

data[i] = bubbles[i].r + " ," + bubbles[i].g + " ," + bubbles[i].diameter; //Każdy element tablicy String
                                                                            //jest tworzony przez łączenie
                                                                            //wartości każdej zmiennej
                                                                            //obiektu Bubble

}

saveStrings( "data/data.txt",data);

}

void mousePressed() {

saveData();

}
```

Ponieważ oryginalny plik danych jest nadpisywany, przy każdym ponownym uruchomieniu szkicu ładowane są nowe wartości. Oto cały przykład w celach informacyjnych.

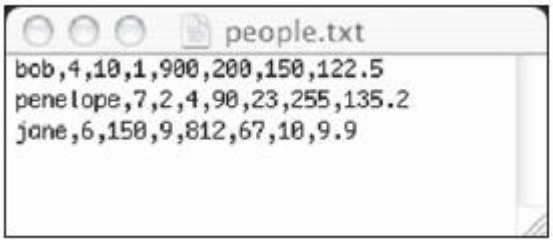
```
// Tablica obiektów Bubble

Bubble[] bubbles;

void setup() {

size(200,200);

smooth();

// Załaduj plik tekstowy jako łańcuch

String[] data = loadStrings(" data.txt " ); //Dane bańki są ładowane w setup ().

// Utwórz w pliku tekstowym tyle obiektów, ile linii
```

```
bubbles = new Bubble[data.length];
// Konwertuj wartości na zmienne i przechodź do konstruktora
for (int i = 0; i < bubbles.length; i + + ) {
float[] values = float(split(data[i], " , " ));
bubbles[i] = new Bubble(values[0],values[1],values[2]);
}
}
void draw() {
background(255);
// Wyświetl i przenieś wszystkie bąbelki
for (int i = 0; i < bubbles.length; i + +) {
bubbles[i].display();
bubbles[i].drift();
// Zmień bąbelki, jeśli mysz się przesunie
if (bubbles[i].rollover(mouseX,mouseY)) {
bubbles[i].change();
}
}
}
// Zapisz nowe dane bańki, gdy mysz jest wciśnięta
void mousePressed() {
saveData();       //Dane bańki są zapisywane w mousePressed ()
}
void saveData() {
// Dla każdej bańki należy zapisać jeden ciąg znaków
String[] data = new String[bubbles.length];
for (int i = 0; i < bubbles.length; i + +) {
// Połącz zmienne bąbelkowe
data[i] = bubbles[i].r + " , " + bubbles[i].g + " , " + bubbles[i].diameter;
}
//Zapisz do pliku
```

```
saveStrings( " data/data.txt ",data);      //Ten sam plik jest zastępowany przez dodanie ścieżki folderu
                                           //„data” do saveStrings ()
}
// klasa Bubble
class Bubble {
float x,y;
float diameter;
float speed;
float r,g;
Bubble(float r_,float g_, float diameter_) {
x = random(width);
y = height;
r = r_;
g = g_;
diameter = diameter_;
}
// Prawda lub Fałsz, jeśli punkt jest wewnątrz okręgu
boolean rollover(int mx, int my) {
if (dist(mx,my,x,y) < diameter/2) {
return true;
} else {
return false;
}
}
// Zmień zmienne bąbelkowe
void change() {
r = constrain(r + random(–10,10),0,255);
g = constrain(g + random(–10,10),0,255);
diameter = constrain(diameter + random(–2,4),4,72);
}
// Wyświetl bańkę
```

```
void display() {
stroke(0);
fill(r,g,255,150);
ellipse(x,y,diameter,diameter);
}
// Bubble dryfuje w górę
void drift() {
y + = random(–3,–0.1);
x + = random(–1,1);
if (y < -diameter*2) {
y = height + diameter*2;
}
}
}
```

Ćwiczenie 18-7: Utwórz szkic, który wizualizuje następujące dane. Możesz dodać i zmienić tekst.



18.4 Przetwarzanie tekstu

Teraz, gdy czujemy się dobrze, wiedząc jak loadStrings () działa dla plików tekstowych przechowywanych lokalnie, możemy zacząć badać, co możemy zrobić z danymi tekstowymi, które pobieramy z innych źródeł, takich jak URL.

String [] lines = loadStrings ("http://www.yahoo.com");

Gdy wysyłasz ścieżkę URL do loadStrings (), otrzymujesz surowe źródło HTML („Hypertext Markup Language") żądanej strony internetowej. Jest to ten sam materiał, który pojawia się po wybraniu „Wyświetl źródło" z opcji menu przeglądarki. Nie musisz być ekspertem HTML, aby śledzić tę sekcję, ale jeśli nie jesteś zaznajomiony z HTML, możesz przeczytać http://en.wikipedia.org/wiki/HTML. W przeciwieństwie do danych rozdzielanych przecinkami z pliku tekstowego, który został specjalnie sformatowany do użycia w szkicu przetwarzania, nie jest praktyczne posiadanie wynikowego surowego kodu HTML przechowywanego w tablicy ciągów (każdy element reprezentujący jedną linię ze źródła). Przekształcenie tablicy w jeden długi ciąg może nieco uprościć. Jak widzieliśmy wcześniej w tym rozdziale, można to osiągnąć za pomocą join().

String onelongstring = join(lines, " ");

Podczas pobierania surowego kodu HTML ze strony internetowej prawdopodobnie nie chcesz całego źródła, ale tylko jego niewielką część. Być może szukasz informacji o pogodzie, wycenę akcji lub nagłówek wiadomości. Możemy skorzystać z funkcji manipulacji String, których się nauczyliśmy - indexOf (), substring () i length () - aby znaleźć fragmenty danych w dużym bloku tekstu. Widzieliśmy wczesny przykład tego w ćwiczeniu 18-2. Weźmy na przykład następujący ciąg:

String stuff = " Number of apples:62. Boy, do I like apples or what! " ;

Powiedzmy, że chcemy wyciągnąć liczbę jabłek z powyższego tekstu. Nasz algorytm byłby następujący:

1. Znajdź koniec podciągów „apple". Zadzwoń na początek.

2. Znajdź pierwszy okres po „applee:". Nazwij to koniec.

3. Utwórz podciąg znaków między początkiem a końcem.

4. Przekonwertuj ciąg na liczbę (jeśli chcemy go użyć jako taką).

W kodzie wygląda to tak:

```
int start = stuff.indexOf( " apples: " ) + 8; // STEP 1      //„Koniec" ciągu można znaleźć, wyszukując
                                                            //indeks tego ciągu i dodając długość do tego
                                                            //indeksu (w tym przypadku 8)

int end = stuff.indexOf( " . " ,start); // STEP 2

String apples = stuff.Substring(start,end); // STEP 3

int apple_no = int(apples); // STEP 4
```

Powyższy kod załatwi sprawę, ale powinniśmy być nieco bardziej ostrożni, aby upewnić się, że nie napotkamy żadnych błędów, jeśli nie znajdziemy podciągów. Możemy dodać sprawdzanie błędów i uogólnić kod na funkcję:

```
// Funkcja zwracająca podciąg między dwoma podciągami

String giveMeTextBetween(String s, String startTag, String endTag) {

String found = " " ;                    //Funkcja zwracająca ciąg znaleziony między dwoma
                                        //łańcuchami. Jeśli początek lub koniec „tagu" nie zostanie
                                        //znaleziony, funkcja zwraca pusty ciąg

// Znajdź indeks znacznika początkowego

int startIndex = s.indexOf(startTag);

//Jeśli nic nie znajdziemy

if (startIndex = = –1) return " " ;

// Przejdź na koniec znacznika początkowego

startIndex + = startTag.length();

// Znajdź indeks znacznika końcowego
```

```
int endIndex = s.indexOf(endTag, startIndex);   // IndexOf () może również przyjąć drugi argument,
                                                //liczbę całkowitą. Ten drugi argument oznacza:
                                                //Znajdź pierwsze wystąpienie ciągu wyszukiwania po
                                                //tym określonym indeksie. Używamy go tutaj, aby
                                                //zapewnić, że indeks końcowy podąża za indeksem
                                                //startowym.

// Jeśli nie znajdziemy tagu końcowego,

if (endIndex = = –1) return " " ;

// Zwróć tekst pomiędzy

return s.substring(startIndex,endIndex);

}
```

Dzięki tej technice jesteśmy gotowi połączyć się ze stroną internetową z poziomu przetwarzania i pobrać dane do wykorzystania w naszych szkicach. Na przykład, możemy przeczytać źródło HTML z http://www.nytimes.com i poszukać dzisiejszych nagłówków, przeszukać http: // fi nance.yahoo.com w celu uzyskania notowań giełdowych, policzyć, ile razy pojawia się słowo „Flickr” na twoim ulubionym blogu i tak dalej. HTML jest jednak brzydkim, przerażającym miejscem niespójnie sformatowane strony, które trudno jest odwrócić i efektywnie przeanalizować. Nie wspominając już o tym, że firmy często zmieniają kod źródłowy stron internetowych, więc każdy przykład, który mogę zrobić podczas pisania tego akapitu, może się zepsuć do czasu przeczytania tego paragrafu. Aby pobrać dane z sieci, kanał XML (Extensible Markup Language) okaże się bardziej niezawodny i łatwiejszy do analizy. W przeciwieństwie do HTML (który ma na celu umożliwienie oglądania treści oczyma człowieka) XML ma na celu umożliwienie przeglądania treści przez komputer i ułatwienie udostępniania danych w różnych systemach. Zapoznamy się z tym, jak działa XML w Sekcji 18.17. Na razie przyjrzyjmy się, jak możemy pobrać pogodę dla dowolnego kodu pocztowego z kanału pogodowego XML Yahoo. Informacje o wszystkich kanałach XML Yahoo można znaleźć tutaj: http://developer.yahoo.com/rss/. Pogoda w formacie XML jest tutaj:

http://xml.weather.yahoo.com/forecastrss?p -10025

Jednym ze sposobów na pobranie danych z kanału pogodowego jest użycie biblioteki Przetwarzanie XML (która ułatwia czytanie z dokumentu XML). Jednak w celu zademonstrowania parsowania String na niższym poziomie, jako ćwiczenie, użyjemy naszych technik scrapingu loadStrings () i ręcznie wyszukasz bity informacji osadzone w źródle XML. Trzeba przyznać, że jest to trochę głupie dążenie, ponieważ XML jest przeznaczony do analizowania bez konieczności uciekania się do tej metodologii. Dla porównania przyjrzymy się przykładowemu przykładowi wykorzystując dwie różne biblioteki XML w sekcjach 18.7 i 18.8. Patrząc na źródło XML z powyższego adresu URL, widzimy, że dzisiejsza temperatura (która w chwili pisania tego tekstu jest 1 sierpnia 2007 r.) Wynosi 88°F— temp = „88”.

<yweather: condition text = "Fair" code = "34" temp = "88" date = "Wed, 01 sierpnia 2007

15:51 EDT ”/?>

Temperatura jest zmienna, ale format XML nie jest i dlatego możemy wywnioskować, że znacznik początkowy dla

nasze wyszukiwanie powinno być:

temp = "

i znacznik końcowy:

”

(tj. pierwszy cytat po tagu początkowym).

Znając znaczniki początkowe i końcowe, możemy użyć metody giveMeTextBetween (), aby wyciągnąć temperaturę.

```
String url = " http://xml.weather.yahoo.com/forecastrss?p = 10003 " ;

String[] lines = loadStrings(url);

// Pozbądź się tablicy, aby przeszukać

// całą stronę

String xml = join(lines, " " );

// Wyszukiwanie temperatury

String tag1 = " temp = \ " " ;          // Cytat w Javie oznacza początek lub koniec łańcucha. Więc
                                        //jak czy uwzględnimy rzeczywisty cytat w ciągu? Odpowiedzią
                                        //jest sekwencja „ucieczki”. (Spotkaliśmy się z tym w ćwiczeniu
                                        //17-8.) Cytat może być zawarty w ciągu za pomocą a ukośnik
                                        //wstecz, a następnie cytat. Na przykład: String q = "This String
                                        //has a quote \"in it";

String tag2 = " \ " " ;

temp = int(giveMeTextBetween

(xml,tag1,tag2));

println(temp);
```

Przykład 18-5 pobiera temperaturę z kanału XML pogody Yahoo i wyświetla go na ekranie. W przykładzie wykorzystano również programowanie obiektowe, umieszczając wszystkie funkcje analizowania ciągu znaków w klasie WeatherGrabber.

Przykład 18-5: Ręczne przetwarzanie danych pogodowych XML Yahoo

```
PFont f;

String[] zips = { " 10003 ", " 21209 ", " 90210 " } ;

int counter = 0;

// Obiekt WeatherGrabber działa za nas!WeatherGrabber wg;

void setup() {

size(200,200);

// Utwórz obiekt WeatherGrabber

wg = new WeatherGrabber(zips[counter]);     // Obiekt WeatherGrabber jest inicjowany kodem
                                            //pocztowym. Dane XML są ładowane i analizowane
                                            //po wywołaniu funkcji requestWeather ().
```

```
                                        //Informacje o pogodzie i temperaturze są pobierane
                                        //z obiektu WeatherGrabber
// Powiedz, żeby poprosił o pogodę
wg.requestWeather();
f = createFont( " Georgia " ,16,true);
}
void draw() {
background(255);
textFont(f);
fill(0);
// Uzyskaj wartości do wyświetlenia
String weather = wg.getWeather();
int temp = wg.getTemp();
// Wyświetl wszystkie rzeczy, które chcemy wyświetlić
text(zips[counter],10,160);
text(weather,10,90);
text(temp,10,40);
text(" Click to change zip. " ,10,180);
// Narysuj mały termometr na podstawie temperatury
stroke(0);
fill(175);
rect(10,50,temp*2,20);
}
void mousePressed() {`          //Dane są ponownie wymagane z nowym kodem pocztowym za
                                //każdym naciśnięciem myszy.
// Zwiększ licznik i uzyskaj pogodę przy następnym kodzie pocztowym
counter = (counter + 1) % zips.length;
wg.setZip(zips[counter]);
wg.requestWeather();
}
// Klasa WeatherGrabber
class WeatherGrabber {
```

```
int temperature = 0;
String weather = " " ;
String zip;
WeatherGrabber(String tempZip) {
zip = tempZip;
}
// Ustaw nowy kod pocztowy
void setZip(String tempZip) {
zip = tempZip;
}
// Zdobądź temperaturę
int getTemp() {
return temperature;
}
// Zdobądź pogodę
String getWeather() {
return weather;
}
// Utwórz rzeczywiste żądanie XML
void requestWeather() {
// Pobierz cały kod źródłowy HTML / XML do tablicy ciągów
// (każda linia jest jednym elementem w tablicy)
String url = " http://xml.weather.yahoo.com/forecastrss?p = " + zip ;
String[] lines = loadStrings(url);
String xml = join(lines, " " ); // Turn array into one long String
// Wyszukiwanie warunków pogodowych
String lookfor = " < yweather:condition text = \ " " ;
String end = " \ " " ;
weather = giveMeTextBetween (xml,lookfor,end);
// Wyszukiwanie temperatury
lookfor = " temp = \ " " ;
```

```
temperature = int(giveMeTextBetween (xml,lookfor,end));

}

// Funkcja, która zwraca podciąg między dwoma podciągami

String giveMeTextBetween(String s, String before, String after) {

String found = " " ;

int start = s.indexOf(before); // Find the index of the beginning tag

if (start = = – 1) return " " ; // If we don't find anything, send back a blank

// String

start + = before.length(); // Move to the end of the beginning tag

int end = s.indexOf(after,start); // Find the index of the end tag

if (end == –1) return ""; // If we don't find the end tag, send back a blank String

return s.substring(start,end); // Return the text in between

}

}
```

Ćwiczenie 18-8: Rozwiń przykład 18-5, aby wyszukać również wysoką i niską temperaturę następnego dnia.

Ćwiczenie 18-9: Spójrz na plik XML „Word of the Day" Yahoo dostępny pod tym adresem URL:

http://xml.education.yahoo.com/rss/wotd/. Użyj ręcznych technik analizowania, aby wyciągnąć Słowo dnia z kanału.

18.5 Analiza tekstu

Ładowanie tekstu z adresu URL nie musi być jedynie ćwiczeniem podczas analizowania małych fragmentów informacji. Przetwarzanie pozwala analizować duże ilości tekstu znalezionego w Internecie z kanałów informacyjnych, artykułów i przemówień do całych książek. Miłym źródłem jest Project Gutenberg (http://www.gutenberg.org/), który udostępnia tysiące tekstów z domeny publicznej. Algorytmy do analizy tekstu zasługują na całą książkę, ale przyjrzymy się jednemu prostemu przykładowi początkującemu. Przykład 18-6 pobiera cały tekst gry Szekspira King Lear i używa podzielonych tokenów (), aby utworzyć tablicę wszystkich słów w grze. Następnie szkic wyświetla słowa pojedynczo, wraz z liczbą, ile razy słowo pojawia się w tekście.

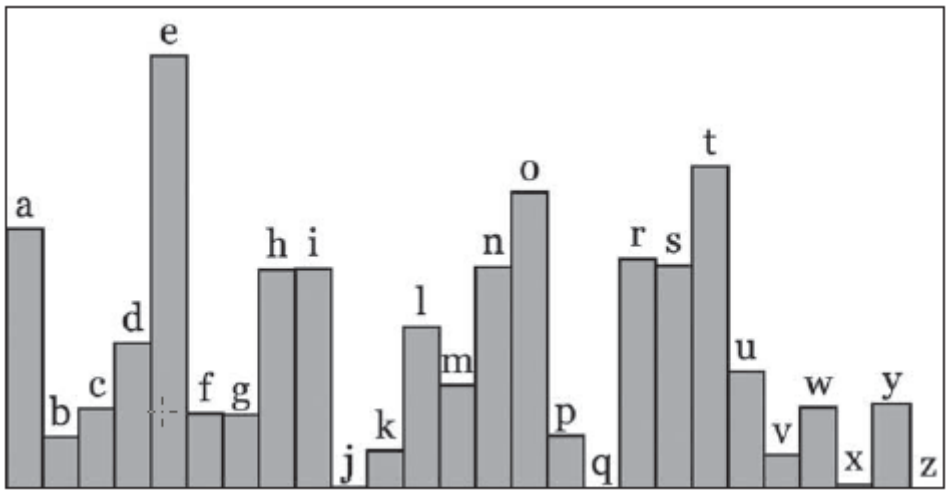
Przykład 18-6: Analiza Króla Leara

```
PFont f; // Zmienna do trzymania czcionki

String[] kinglear; // Tablica do przechowywania całego tekstu

int counter = 0; // Gdzie jesteśmy w tekście

// Będziemy używać spacji i interpunkcji jako ograniczników

String delimiters = " ,.?!;: " ;       // Każdy znak interpunkcyjny  jest używana jako separator.
```

```
void setup() {

size(200,200);

//  Ładowanie czcionki

f = loadFont( " Georgia-Bold-16.vlw " );

// Załaduj króla Leara w szereg ciągów

String url = " http://www.gutenberg.org/dirs/etext97/1ws3310.txt " ;

String[] rawtext = loadStrings(url);

// Połącz dużą tablicę razem jako jeden długi ciąg

String everything = join(rawtext, " " );

// Podziel tablicę na słowa za pomocą dowolnego separatora

kinglear = splitTokens(everything,delimiters);

frameRate(5);   // Wszystkie wiersze w King Lear są najpierw połączone jako jeden duży ciąg, a
                //następnie podzielone na szereg pojedynczych słów. Zwróć uwagę na użycie
                //splitTokens (), ponieważ używamy spacji i znaków interpunkcyjnych jako
                //ograniczników.

}

void draw() {

background(255);

// Wybierz jedno słowo od Króla Leara

String theword = kinglear[counter];

// Policz, ile razy to słowo pojawia się w King Lear      // Ta pętla zlicza liczbę wystąpień aktualnie
                                                          //wyświetlanego słowa.

int total = 0;

for (int i = 0; i < kinglear.length; i + + ) {

if (theword.equals(kinglear[i])) {

total + + ;

}

}

// Wyświetl tekst i całkowity czas pojawienia się tego słowa

textFont(f);

fill(0);

text(theword,10,90);
```

```
text(total,10,110);

stroke(0);

fill(175);

rect(10,50,total/4,20);

// Przejdź do następnego słowa

counter = (counter + 1) % kinglear.length;

}
```

Ćwiczenie 18-10: Policz, ile razy każda litera alfabetu pojawia się w King Lear Wizualizuj te liczby. Oto jedna z możliwości (powinieneś być bardziej kreatywny). Zauważ, że ten szkic będzie wymagał użycia funkcji charAt ().



18.6 Żądania asynchroniczne

Jak widzieliśmy, funkcja loadStrings () może być używana do pobierania surowych danych ze stron internetowych. Niemniej jednak, jeśli szkic nie wymaga tylko jednokrotnego załadowania danych podczas instalacji (), możesz mieć problem. Na przykład rozważ szkic, który pobiera cenę zapasów AAPL z kanału XML co 5 minut. Za każdym razem, gdy wywoływany jest loadStrings (), szkic zostanie wstrzymany w oczekiwaniu na otrzymanie danych. Każda animacja będzie się jąkać. Dzieje się tak, ponieważ loadStrings () jest funkcją „blokującą", innymi słowy, szkic będzie siedział i czekał w tym wierszu kodu, aż loadStrings () zakończy swoje zadanie. Z lokalnym plikiem tekstowym jest to niezwykle szybkie. Niemniej jednak prośba o stronę internetową (zwaną „żądaniem HTTP") jest asynchroniczna, co oznacza, że serwer WWW może poświęcić trochę czasu na otrzymanie żądanych danych. Kto wie, jak długo zajmie loadStrings ()? Nikt, jesteś na łasce serwera! Biblioteka simpleML (dostępna do pobrania na stronie internetowej tej książki: http: //www.learniningprocessing. Com / simpleML /) omawia ten problem, wykonując równolegle te żądania HTTP do serwerów WWW bez wstrzymywania szkicu przetwarzania, umożliwiając szkic wielozadaniowość i kontynuuj animację, gdy trwa pobieranie danych. Biblioteka działa w podobny sposób jak biblioteka wideo przetwarzania, którą zapoznaliśmy w części 16. Aby pobrać stronę internetową, należy utworzyć instancję obiektu HTMLRequest, przekazując odniesienie do samego szkicu (to) i adres URL, którego chcesz zażądać.

```
HTMLRequest req = new HTMLRequest (this, „http://www.yahoo.com");
```

Żądanie nie rozpocznie się jednak dopóki nie wywołasz makeRequest ().

req.makeRequest (); Wreszcie, aby otrzymać dane, musisz napisać funkcję o nazwie netEvent (). Ta funkcja zostanie wywołana natychmiast po zakończeniu żądania i udostępnieniu danych. Ta funkcja jest kolejnym przykładem funkcji wywołania zwrotnego, takiej samej jak funkcja captureEvent () lub mousePressed (). W tym przypadku, zamiast wyzwalania zdarzenia, gdy użytkownik kliknie myszą lub obraz z kamery jest dostępny, zdarzenie jest wyzwalane, gdy HTML żąda fiasku (lub XML, jak zobaczymy za chwilę). Źródło HTML strony internetowej jest zwracane jako łańcuch za pośrednictwem funkcji readRawSource ().

```
void netEvent (HTMLRequest ml) {

String html = ml.readRawSource ();

println (html);

}
```

Przykład 18-7 pobiera stronę główną Yahoo co dziesięć sekund przy użyciu biblioteki simpleML. Wizualizacja tutaj jest dowolna (linie są kolorowe zgodnie ze znakami w źródle HTML Yahoo); ważne jest jednak, aby zauważyć, że animacja nigdy nie zatrzymuje się w oczekiwaniu na nadejście danych.

Przykład 18-7: Ładowanie adresu URL za pomocą simpleML

```
import simpleML.*;

// Obiekt żądania z biblioteki

HTMLRequest htmlRequest;

Timer timer = new Timer(5000);

String html = " " ; // Ciąg do przechowywania danych z żądania

int counter = 0; // Licznik do animacji prostokąta w oknie

int back = 255; // Jasność tła

void setup() {

size(200,200);

// Utwórz i wykonaj żądanie asynchroniczne

htmlRequest = new HTMLRequest(this, " http://www.yahoo.com " );

htmlRequest.makeRequest();    // Obiekt HTML Request, aby zażądać źródła z adresu URL

timer.start();

background(0);

}

void draw() {

background(back);

// Co 5 sekund zgłoś nową prośbę
```

```
if (timer.isFinished()) {
htmlRequest.makeRequest();    // Żądanie jest wysyłane co 5 sekund. Dane nie są otrzymywane tutaj,
                              //ale to tylko prośba.
println( "Making request! ");
timer.start();
}
// Narysuj linie z kolorami opartymi na znakach z pobranych danych
for (int i = 0; i < width; i + + ) {
if (i < html.length()) {
int c = html.charAt(i);
stroke(c,150);
line(i,0,i,height);
}
}
// Animuj prostokąt i przyciemniony prostokąt
fill(255);
noStroke();
rect(counter,0,10,height);
counter = (counter + 1) % width;
back = constrain(back – 1,0,255);
}
// Po zakończeniu żądania
void netEvent(HTMLRequest ml) {       // Dane są odbierane w funkcji netEvent (), która jest
                                      //automatycznie wywoływana, gdy dane są gotowe.
html = ml.readRawSource(); // Read the raw data
back = 255; // Reset background
println( "Request completed! "); // Print message
}
// Klasa Timer z części 10
class Timer {
int savedTime;
boolean running = false;
```

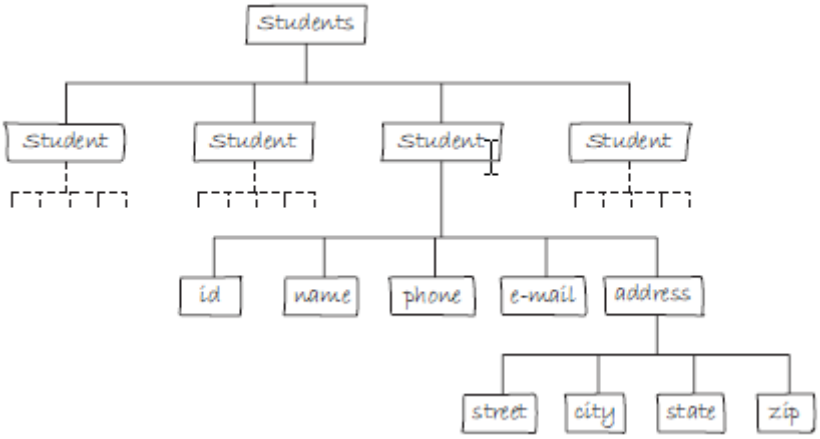
```
int totalTime;

Timer(int tempTotalTime) {

totalTime = tempTotalTime;

}

void start() {

running = true;

savedTime = millis();

}

boolean isFinished() {

int passedTime = millis() – savedTime;

if (running & & passedTime > totalTime) {

running = false;

return true;

} else {

return false;

}

}

}
```

18,7 Początkujący XML

Przykłady w sekcji 18.4 pokazały proces ręcznego przeszukiwania tekstu w poszukiwaniu pojedynczych danych. Wprawdzie ręczne analizowanie surowych danych XML z

http://xml.weather.yahoo.com/forecastrss?p = 10003

nie była najbardziej skuteczną strategią. Tak, jeśli potrzebujesz informacji z dziwnie uformowanej strony HTML, wymagane są te techniki ręczne. Ponieważ jednak XML został zaprojektowany w celu ułatwienia udostępniania danych w różnych systemach, można go analizować bez ręcznego wyszukiwania za pomocą biblioteki analizującej XML. XML organizuje informacje w strukturze drzewa. Wyobraźmy sobie listę uczniów. Każdy uczeń ma numer identyfikacyjny, imię, adres, e-mail i numer telefonu. Adres każdego ucznia ma miasto, stan i kod pocztowy. Drzewo XML dla tego zestawu danych może wyglądać jak rysunek

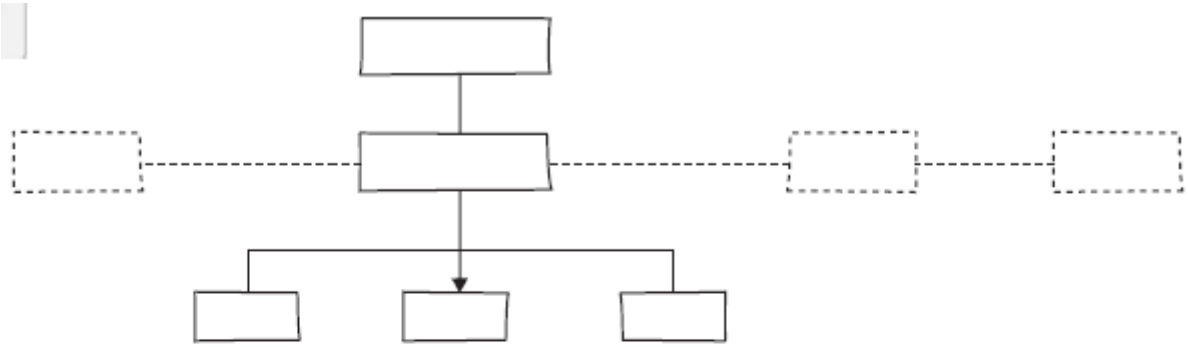


Samo źródło XML (z dwoma studentami na liście) to:

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
      <student>
            <id>001</id>
            <name>Daniel Shiffman</name>
            <phone>555-555-5555</phone>
            <email>daniel@shiffman.net</email>
            <address>
                  <street>123 Processing Way</street>
                  <city>Loops</city>
                  <state>New York</state>
                  <zip>01234</zip>
            </address>
      </student>


    <student>
            <id>002</id>
            <name>Zoog</name>
            <phone>555-555-5555</phone>
            <email>zoog@planetzoron.uni</email>
            <address>
                  <street>45.3 Nebula 5</street>
                  <city>Boolean City</city>
                  <state>Booles</state>
                  <zip>12358</zip>
            </address>
    </student>
</students>
```

Zwróć uwagę na podobieństwa do programowania obiektowego. Możemy pomyśleć o drzewie XML w następujących terminach. Dokument XML reprezentuje szereg obiektów uczniów. Każdy obiekt ucznia ma wiele elementów informacji, identyfikatora, nazwiska, numeru telefonu, adresu e-mail i adresu pocztowego. Adres pocztowy jest również obiektem, który zawiera wiele elementów danych, takich jak ulica, miasto, stan i kod pocztowy.

Ćwiczenie 18-11: Powróć do przykładu „Bańka” 18-3. Zaprojektuj strukturę drzewa XML dla obiektów Bubble. Utwórz diagram drzewa i wypisz źródło XML. (Użyj pustego diagramu i wypełnij puste pola poniżej.)

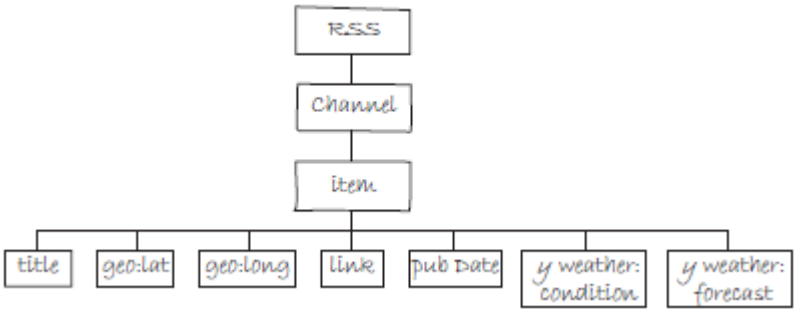


```
<?xml version="1.0"?>
<_______>
  <bubble>
    <_______>40     </_______>
    <_______>100  </_______>
    <_______>255  </green>
  <_______>
</bubbles>
```

Wracając do przykładu pogody, możemy teraz poczuć sens danych XML Yahoo z warunkami drzewa w drzewie. Oto surowe źródło XML. (Uwaga: edytowałem go w celu uproszczenia).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<rss version="2.0" xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0">
      <channel>
    <item>
      <title>Conditions for New York, NY at 3:51 pm EST</title>
      <geo:lat>40.67</geo:lat>
      <geo:long>-73.94</geo:long>
      <link>http://xml.weather.yahoo.com/forecast/USNY0996_f.html</link>
      <pubDate>Mon, 20 Feb 2006 3:51 pm EST</pubDate>
      <yweather:condition text="Fair" code="34" temp="35" date="Mon, 20 Feb 2006
        3:51 pm EST"/>
      <yweather:forecast day="Mon" date="20 Feb 2006" low="25" high="37"
        text="Clear" code="31"/>
    </item>
      </channel>
</rss>
```

Dane są mapowane w strukturze drzewa pokazanej na rysunku



Być może zastanawiasz się, o co chodzi w „RSS" najwyższego poziomu. Dane pogodowe XML Yahoo są dostarczane w formacie RSS. RSS oznacza „Really Simple Syndication" i jest znormalizowanym formatem XML do syndykacji treści internetowych (takich jak artykuły prasowe itp.). Możesz przeczytać więcej o RSS na wikipedii:

http://en.wikipedia.org/wiki/RSS.

Teraz, gdy mamy uchwyt na strukturę drzewa, powinniśmy spojrzeć na specyfikacje wewnątrz tej struktury. Z wyjątkiem pierwszej linii (która po prostu wskazuje, że ta strona jest sformatowana w formacie XML), ten dokument XML zawiera zagnieżdżoną listę elementów, każdy ze znacznikiem początkowym, czyli? kanał? i znacznik końcowy, czyli? / kanał? . Niektóre z tych elementów zawierają treści między tagami:

<title> Warunki dla Nowego Jorku, NY o 15:51 czasu EST? </title>

a niektóre mają atrybuty (sformatowane przez nazwę atrybutu równą wartości atrybutu w cudzysłowie):

<yweather: forecast day = "Mon" date = "20 lutego 2006" low = "25" high = "37" text = "Clear" code = "31" />

Dostępnych jest kilka bibliotek analizujących XML do przetworzenia, aw tym rozdziale omówimy dwa. Pierwsza, zaprojektowana dla tej książki, jest najprostsza (ale ma najmniejszą liczbę funkcji): biblioteka simpleML.

simpleML może wykonywać następujące trzy zadania:

• Pobierz tekst z jednego elementu XML jako ciąg.

• Pobierz tekst z jednego atrybutu elementu XML jako ciąg.

• Pobierz tekst z wielu elementów XML (z tym samym znacznikiem) jako tablicę ciągów.

Żądania danych XML są tworzone za pomocą obiektu XMLRequest.

XMLRequest req = new XMLRequest (this, "http://xml.weather.yahoo.com / forecastrss? p = 10003 ");

Ponownie, żądanie nie rozpocznie się, dopóki nie wywołasz makeRequest ().

req.makeRequest ();

Podobnie jak w przypadku żądania HTML, aby otrzymać dane z żądania XML, musisz zaimplementować netEvent (), tym razem z obiektem XMLRequest jako argumentem. Funkcja getElementText () zwraca zawartość z pojedynczego elementu XML (nazwa tego element musi zostać przekazany do funkcji). Dla atrybutu użyj getElementAttributeText () z nazwą elementu, po której następuje nazwa atrybutu.

Na przykład z następującymi danymi XML:

<geo: lat> 40.67 </ geo: lat>

<yweather: forecast day = "Mon" date = "20 lutego 2006" low = "25" high = "37"

text = "Clear" code = "31" />

Kod do pobierania danych to:

```
void netEvent (XMLRequest ml) {

// Pobieranie tekstu jednego elementu

String lat = ml.getElementText ("geo: lat");      // Treść elementu XML jest odzyskiwana poprzez
                                                  //przekazanie nazwy elementu (w tym przypadku
                                                  //„geo: lat”) do funkcji getElementText ()

// Pobieranie tekstu jednego atrybutu z jednego elementu

String temperature = ml.getElementAttributeText („yweather: forecast”, „high”);

println („Współrzędne to:” + lat);      // Zawartość atrybutu elementu XML jest pobierana przez
                                        //przekazanie w nazwie elementu (w tym przypadku
                                        //„yweather: forecast”) oraz nazwa atrybutu (w tym
                                        //przypadku „wysoka”) do funkcji getElementAttrributeText ().

println („Wysoka temperatura to:

„+ temperatura);

}
```

Metoda getElementArray () może być również wywołana w celu pobrania elementów XML, które pojawiają się wiele razy. Zwraca tablicę ciągów, jeden ciąg dla każdego elementu w dokumencie XML. Poniższy przykład pobiera wszystkie nagłówki z Yahoo's Top Stories XML Feed.

Przykład 18-8: Ładowanie XML z simpleML

```
import simpleML.*;

XMLRequest xmlRequest;

void setup() {

size(200,200);

// Tworzenie i uruchamianie żądania

xmlRequest = new XMLRequest(this, " http://rss.news.yahoo.com/rss/topstories " );

xmlRequest.makeRequest();

}                                       // Tablicę elementów XML można pobrać za pomocą getElementArray.
                                        //Działa to tylko w przypadku elementów o tej samej nazwie, które
                                        //pojawiają się wielokrotnie w dokumencie XML.

void draw() {
```

```
noLoop();// Nothing to see here
}
// Po zakończeniu żądania
void netEvent(XMLRequest ml) {
// Pobieranie tablicy wszystkich elementów XML wewnątrz znaczników „title *”
String[] headlines = ml.getElementArray( " title " );
for (int i = 0; i < headlines.length; i + + ) {
println(headlines[i]);
}
}
```

Ćwiczenie 18-12: Wizualizuj dane pogodowe Yahoo za pomocą simpleML. Poniżej znajduje się część kodu, który pomoże Ci zacząć. To jest XML na wypadek, gdybyś zapomniał: <yweather: condition text = „Fair” code = „34” temp = „35” date = „Mon, 20 Feb 2006 3:51 pm EST” /

```
import simpleML.*;
// Variables for temperature and weather
int temperature = 0;
String weather = " " ;
void setup() {
// FILL THIS IN HOWEVER YOU LIKE!
}
void draw() {
// FILL THIS IN HOWEVER YOU LIKE!
}
// Function that makes the weather request with a Zip Code
void getWeather(String zip) {
String url = " http://xml.weather.yahoo.com/
forecastrss?p = " + zip ;
XMLRequest req = new XMLRequest(this,url);
req.makeRequest();
}
void netEvent(XMLRequest ml) {
// Get the specifi c XML content we want
```

temperature = int(ml._____(" _____ " , " _____ "));

weather = ml._____(" _____ " , " _____ ");

}

18.8 Korzystanie z biblioteki przetwarzania XML

Funkcjonalność biblioteki simpleML, choć łatwa w użyciu, jest dość ograniczona. Jeśli chcesz tworzyć własne dokumenty XML lub analizować wiele elementów dokumentu za pomocą niestandardowego algorytmu, nie masz szczęścia. Dla bardziej wyrafinowanej funkcjonalności XML istnieją dwie opcje. Pierwszą, bardziej zaawansowaną opcją jest proXML (http://www.texone.org/proxml/) stworzony przez Christiana Riekoffa. Podczas gdy krzywa uczenia się jest nieco bardziej stroma, masz bezpośredni dostęp do struktury drzewa XML i możesz czytać i pisać dokumenty XML. Druga opcja, którą omówimy w tym rozdziale, to wbudowana biblioteka XML przetwarzania. import processing.xml. *; Po zaimportowaniu biblioteki pierwszym krokiem jest utworzenie obiektu XMLElement. Obiekt jest ładowany z danymi z dokumentów XML (przechowywane lokalnie lub w sieci). Konstruktor wymaga dwóch argumentów, „this” i nazwy pliku lub ścieżki URL dla dokumentu XML.

String url = "xmldocument.xml";

XMLElement xml = new XMLElement (this, url);

W przeciwieństwie do simpleML ta biblioteka XML wstrzymuje szkic i czeka na załadowanie dokumentu. W przypadku asynchronicznego podejścia do parsowania XML należy użyć proXML. Obiekt XMLElement reprezentuje jeden element drzewa XML. Gdy dokument jest ładowany po raz pierwszy, ten obiekt elementu jest zawsze elementem głównym. simpleML wykonał zadanie przemierzenia drzewa i znalezienia dla nas odpowiednich informacji. Dzięki bibliotece Processing XML musimy wykonać tę pracę sami. Mimo że jest to bardziej złożone, mamy większą kontrolę nad tym, jak szukamy i czego szukamy. Wracając do rysunku 18.13, możemy znaleźć temperaturę za pomocą następującej ścieżki:

1. Elementem głównym drzewa jest „RSS. ”

2. „RSS” ma element podrzędny o nazwie „Kanał. ”

3. 13. dziecko „Kanału” to „element. ”(Diagram jest uproszczony, aby pokazać tylko jedno dziecko kanału.)

4. Szóste dziecko „przedmiotu” to „yweather: warunek. ”

5. Temperatura jest zapisywana w „yweather: condition” jako atrybut „temp. ”

Dostęp do potomków elementu uzyskuje się za pomocą indeksu (zaczynającego się od zera, takiego samego jak tablica) przekazanego do funkcji getChild (). Treść elementu jest pobierana za pomocą getContent (), a atrybuty są odczytywane jako liczby - getIntAttribute (), getFloatAttribute () - lub tekst - getStringAttribute ().

// Dostęp do pierwszego elementu potomnego elementu głównego

XMLElement channel = xml.getChild (0);

Po wykonaniu kroków od 1 do 5 opisanych powyżej za pomocą drzewa XML mamy:

XMLElement xml = new XMLElement (this, url);

XMLElement channel = xml.getChild (0);

XMLElement item = channel.getChild (12); //Trzynastym dzieckiem elementu jest indeks # 12

XMLElement condition = item.getChild (5);

temp = condition.getIntAttribute ("temp");

Inne przydatne funkcje, które mogą wywoływać obiekty XMLElement to:

• getChildCount () - zwraca całkowitą liczbę dzieci XMLElement.

• getChildren () - zwraca wszystkie dzieci jako tablicę XMLElements.

W przykładzie 18-3 użyliśmy serii wartości rozdzielonych przecinkami w pliku tekstowym do przechowywania informacji związanych z obiektami Bubble. Dokument XML może być również używany w ten sam sposób. Oto możliwe rozwiązanie ćwiczenia 18-11, drzewa XML obiektów Bubble. (Zauważ, że to rozwiązanie wykorzystuje atrybuty elementów dla kolorów czerwonego i zielonego; nie był to format podany w ćwiczeniu 18-11, ponieważ nie poznaliśmy jeszcze atrybutów).

< ?xml version = " 1.0 " ? >

< bubbles > //Elementem głównym są „bąbelki”, które ma troje dzieci.

< bubble >

< diameter > 40 </diameter >

< color red = " 75 " green = " 255 " />

</bubble >

< bubble >

< diameter> 20 < /diameter >

< color red = " 255 " green = " 75" / >

< /bubble >

<buble> //Każda „bańka” dziecka ma dwoje dzieci, „średnicę” i „kolor”. Element „kolor” //ma dwa atrybuty: „czerwony” i „zielony”.

< diameter > 80 < /diameter >

< color red = " 100 " green = " 150 " / >

< /bubble >

< /bubbles>

Możemy użyć getChildren (), aby otrzymać tablicę elementów „Bubble” i uczynić obiekt Bubble z każdego z nich. Oto przykład (który używa identycznej klasy Bubble z wcześniejszej). Nowe elementy są pogrubione.

Przykład 18-9: Korzystanie z biblioteki XML przetwarzania

import processing.xml.*;

// Tablica obiektów Bubble

```
Bubble[] bubbles;

void setup() {

size(200,200);

smooth();

// Załaduj dokument XML

XMLElement xml = new XMLElement(this, " bubbles.xml " );

// Pobierz całkowitą liczbę bąbelków

int totalBubbles = xml.getChildCount(); //Pobieranie całkowitej liczby obiektów Bubble za pomocą
                                        //getChildCount ().

// Utwórz tablicę o tym samym rozmiarze

bubbles = new Bubble[totalBubbles];

//  Pobierz wszystkie elementy potomne

XMLElement[] children = xml.getChildren();

for (int i = 0; i < children.length; i + + ) {

//  Średnica to dziecko 0

XMLElement diameterElement = children[i].getChild(0);

int diameter = int(diameterElement.getContent());

//  Kolor to dziecko 1                          //Średnica jest zawartością pierwszego
                                                //elementu, natomiast czerwony i zielony są
                                                //atrybutami drugiego

XMLElement colorElement = children[i].getChild(1);

int r = colorElement.getIntAttribute( "red");

int g = colorElement.getIntAttribute( "green");

// Zrób nowy obiekt Bubble z wartościami z dokumentu XML

bubbles[i] = new Bubble(r,g,diameter);

}

}

void draw() {

background(100);

// Wyświetl i przenieś wszystkie bąbelki

for (int i = 0; i < bubbles.length; i + + ) {

bubbles[i].display();
```

```
bubbles[i].drift();
}
}
```

Ćwiczenie 18-13: Użyj następującego dokumentu XML, aby zainicjować tablicę obiektów. Zaprojektuj obiekty tak, aby używały wszystkich wartości w każdym elemencie XML. (Nie zapomnij przepisać dokumentu XML, aby zawierał więcej lub mniej danych).

```
<?xml version="1.0"?>
<blobs>
  <blob>
    <location x="99" y="192"/>
    <speed x="−0.88238335" y="2.2704291"/>
    <size w="38" h="10"/>
  </blob>
  <blob>
    <location x="97" y="14"/>
    <speed x="2.8775783" y="2.9483867"/>
    <size w="81" h="43"/>
  </blob>
  <blob>
    <location x="159" y="193"/>
    <speed x="-1.2341062" y="0.44016743"/>
    <size w="19" h="95"/>
  </blob>
  <blob>

    <location x="102" y="53"/>
    <speed x="0.8000488" y="-2.2791147"/>
    <size w="25" h="95"/>
  </blob>
  <blob>
    <location x="152" y="181"/>
    <speed x="1.9928784" y="-2.9540048"/>
    <size w="74" h="19"/>
  </blob>
</blobs>
```

18.9 Interfejs API Yahoo

Ładowanie dokumentów HTML i XML jest wygodne do wyciągania informacji z sieci, jednak w przypadku bardziej zaawansowanych aplikacji wiele witryn posiada API. Interfejs API (Application

Programming Interface) to interfejs, za pośrednictwem którego jedna aplikacja może uzyskać dostęp do usług innego. Istnieje wiele interfejsów API, których można używać z przetwarzaniem. Możesz przejrzeć sekcję „Dane / protokoły” na stronie internetowej Biblioteki Processing

http: /processing.org/reference/libraries/index.html)

dla niektórych pomysłów. W tej sekcji przyjrzymy się przykładowi, który wykonuje wyszukiwanie w sieci za pomocą interfejsu API Yahoo. Chociaż można uzyskać bezpośredni dostęp do interfejsu API Yahoo, utworzyłem bibliotekę Przetwarzania, która sprawia, że jest nieco prostsza (a także umożliwia przeprowadzanie wyszukiwania asynchronicznie i nie powoduje wstrzymania szkicu). Będziesz musiał pobrać zarówno moją bibliotekę przetwarzania, jak i pliki interfejsu API Yahoo (znane są jako SDK: „Software Development Kit”). Instrukcje na ten temat można znaleźć na stronie biblioteki:

http://www.learningprocessing.com/libraries/yahoo/

Po pobraniu plików musisz najpierw uzyskać klucz API Yahoo. W wielu przypadkach firmy będą wymagać rejestracji i uzyskania klucza przed uzyskaniem dostępu do ich interfejsu API. W ten sposób mogą śledzić twoje użytkowanie i upewnić się, że nie masz nic złego. Jest to niewielka cena za bezpłatny programowy dostęp do funkcji Yahoo. Możesz zarejestrować się na identyfikator tutaj: https://developer.yahoo.com/wsregapp/index.php

Gdy masz klucz, jesteś gotowy do pracy. Biblioteka działa w podobny sposób jak simpleML. Tworzysz obiekt YahooSearch i wywołujesz funkcję search (). Po zakończeniu wyszukiwania pojawi się wywołanie zwrotne zdarzenia: searchEvent (). Możesz poprosić o informacje o wynikach wyszukiwania, takie jak adresy URL, tytuły lub podsumowania (wszystkie dostępne jako tablice ciągów).

```
import pyahoo.*;

YahooSearch yahoo;      //Utwórz obiekt YahooSearch. Musisz przekazać klucz API podany przez Yahoo.

void setup() {

size(400,400);

// Utwórz obiekt wyszukiwania, podaj klucz

yahoo = new YahooSearch(this, "YOUR API KEY HERE ");

}

void mousePressed() {

yahoo.search( "processing.org");        //Wyszukaj ciąg. Domyślnie otrzymasz 10 wyników. Jeśli
                                        //chcesz więcej (lub mniej), możesz poprosić o określoną
                                        //liczbę, mówiąc:

yahoo.search („processing. org”, 30)

}

void draw() {

noLoop();

}

// Po zakończeniu wyszukiwania
```

```
void searchEvent(YahooSearch yahoo) {

// Uzyskaj tytuły i adresy URL

String[] titles = yahoo.getTitles();

String[] urls = yahoo.getUrls();  // Wyniki wyszukiwania pojawiają się jako tablica ciągów. Możesz
                                  //także uzyskać podsumowania za pomocą getSummaries ().

for (int i = 0; i < titles.length; i + + ) {

println( "_____");

println(titles[i]);

println(urls[i]);

}

}
```

Biblioteka może być użyta do przeprowadzenia prostej wizualizacji. Poniższy przykład wyszukuje pięć nazw i rysuje okrąg dla każdego z nich (o wielkości powiązanej z całkowitą liczbą dostępnych wyników).

Przykład 18-11: Wizualizacja wyszukiwania Yahoo

```
import pyahoo.*;

YahooSearch yahoo;

PFont f;

// Nazwy do wyszukiwania, tablica obiektów „Bubble”

String[] names = { " Aliki","Cleopatra","Penelope","Daniel","Peter" } ;

Bubble[] bubbles = new Bubble[names.length];

int searchCount = 0;

void setup() {

size(500,300);

yahoo = new YahooSearch(this, "YOUR APPI HERE ");

f = loadFont( "Georgia-20.vlw");

textFont(f);

smooth();

// Wyszukaj wszystkie nazwiska

for (int i = 0; i < names.length; i + + ) {

yahoo.search(names[i]);          // Funkcja search () jest wywoływana dla każdej nazwy w tablicy.

}

}
```

```
void draw() {
background(255);
// Pokaż wszystkie bąbelki
for (int i = 0; i searchCount; i + + ) {
bubbles[i].display();
}
}
// Wyszukiwania przychodzą pojedynczo
void searchEvent(YahooSearch yahoo) {
// Całkowita # liczba wyników dla każdego wyszukiwania
int total = yahoo.getTotalResultsAvailable();   // getTotalResultsAvailable () zwraca całkowitą liczbę
                                                //stron internetowych znalezionych przez yahoo
                                                //zawierających wyszukiwane wyrażenie. Liczby te
                                                //mogą być tak duże, jak są zmniejszane
// Skaluj numer, aby można go było wyświetlić
float r = sqrt(total)/75;
// Utwórz nowy obiekt bąbelkowyt
Bubble b = new Bubble(yahoo.getSearchString(), r,50 + searchCount*100,height/2);
bubbles[searchCount] = b;                       //Dane wyszukiwania służą do utworzenia obiektu
                                                //bąbelkowego do wizualizacji
searchCount + + ;
}
// Prosta klasa „Bubble” do reprezentowania każdego wyszukiwania
class Bubble {
String search;
float x,y,r;
Bubble(String search_, float r_, float x_, float y_) {
search = search_;
r = r_;
x = x_;
y = y_;
}
```

```
void display() {

stroke(0);

fill(0,50);

ellipse(x, y, r, r);

textAlign(CENTER);

fill(0);

text(search,x,y);

}
```

18.10 Piaskownica

Podczas lokalnego uruchamiania programu ze środowiska programistycznego przetwarzania możesz swobodnie dotrzeć do portów i sieci. Jednak podczas uruchamiania programu w przeglądarce internetowej istnieją pewne wymagania dotyczące bezpieczeństwa, podobnie jak wiemy o połączeniu z kamerą wideo w Części 16. Dozwolone jest żądanie adresu URL w tej samej domenie. Na przykład, jeśli Twój aplet jest przechowywany na twojej stronie internetowej:

http://www.myrockindomain.com jesteś w porządku z:

```
//Działa w przeglądarce browserString [] lines = loadStrings ("http://www.myrockindomain.com/data.html");
```

Jeśli jednak poprosisz o adres URL, nie masz szczęścia.

```
// Nie działa w przeglądarce

String [] lines = loadStrings ("http://www.yourkookoodomain.com");
```

Rozwiązaniem tego problemu jest utworzenie skryptu proxy, który mieszka na serwerze z twoim apletem, łączy się z zewnętrznym adresem URL i przekazuje te informacje z powrotem do istoty apletu, próbujesz przekonać swój aplet do myślenia, że pobiera tylko informacje lokalne , Innym rozwiązaniem jest „podpisanie" swojego apletu. Podpisywanie apletu to proces mówienia „Witaj, nazywam się Daniel Shiff i stworzyłem ten aplet. Jeśli ufasz, że powiem „tak", aby ten aplet mógł uzyskać dostęp do zasobów, może to zrobić zwykle nie ma dostępu. „Ponownie, jeśli dopiero tworzysz szkice lokalnie na swoim komputerze w trakcie środowisko Processing nie będzie miało problemu. Jeśli chcesz, aby Twój aplet działał na serwerze internetowym i musisz wysyłać żądania stron na tym serwerze, (http://www.learningprocessing.com/sandbox/) dla przykładowego skryptu proxy PHP, a także wskazówki, jak podpisać aplety.

Strumienie danych

19.1 Synchroniczne a asynchroniczne

W części 18 przyjrzeliśmy się, jak możemy zażądać surowego źródła adresu URL za pomocą loadStrings (), biblioteki simpleML lub biblioteki XML. Składasz wniosek, siadasz i czekasz na wyniki. Być może zauważyłeś, że ten proces nie następuje natychmiastowo. Czasami program może zatrzymać się na kilka sekund (a nawet minut) podczas ładowania strony internetowej lub dokumentu XML. Wynika to z długości czasu wymaganego na wykonanie przetwarzania za kulisami - żądania HTTP. HTTP oznacza „Hypertext Transfer Protocol", protokół żądania / odpowiedzi do przesyłania informacji w sieci WWW. Zastanówmy się przez chwilę, co rozumiemy przez „żądanie / odpowiedź. „Być może budzisz się pewnego ranka i myślisz sobie, że porządek ma urlop, powiedzmy w Toskanii. Włączasz komputer, uruchamiasz przeglądarkę internetową, wpisujesz www.google.com w pasku adresu i wpisujesz „romantyczna willa w Toskanii". Państwa klient, złożył wniosek, a zadaniem serwera Google, jest udzielenie odpowiedzi.

Klient :

Aby się przedstawić, jestem Firefox, przeglądarka internetowa i mam prośbę. Zastanawiałem się, czy możesz być tak uprzejmy, aby wysłać mi swoją stronę o willach wakacyjnych w Toskanii?

[Pauza dramatyczna]

Serwer:

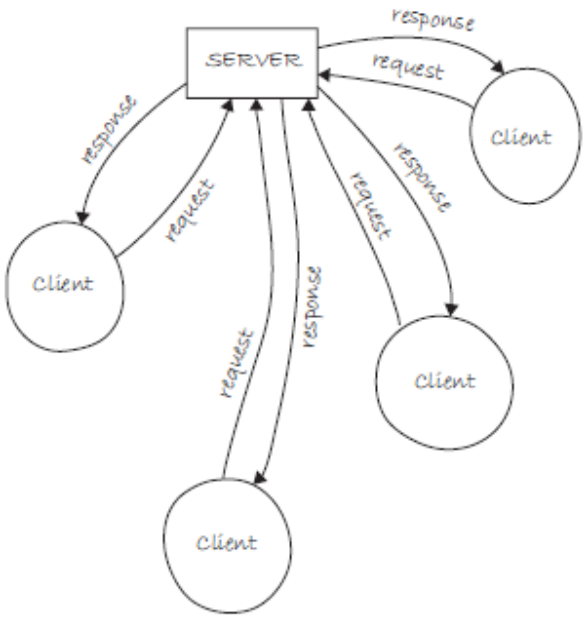
Jasne, nie ma problemu, oto moja odpowiedź. To naprawdę dużo bajtów, ale jeśli przeczytasz go jako html, zobaczysz, że jest to ładnie sformatowana strona o wakacjach w Toskanii. Cieszyć się! Och, możesz dać mi znać, że wszystko było w porządku?

Klient:

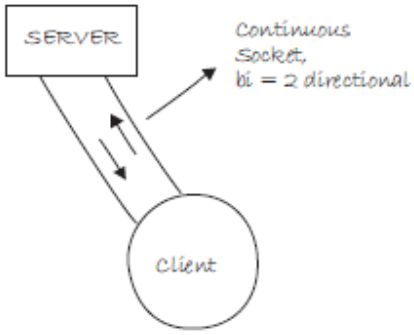
Mam to, dzięki!

[Klient i serwer podają sobie ręce.]

Powyższy proces nazywany jest asynchronicznym żądaniem i odpowiedzią, dwukierunkową komunikacją między serwerem a klientem, gdzie serwer odpowiada w swoim czasie wolnym. Połączenie jest ustanawiane tymczasowo w celu przeniesienia strony internetowej, po której jest szybko zamykane. Jeśli chodzi o wysyłanie informacji przez sieć światową, ta metodologia jest całkowicie wystarczająca. Niemniej jednak wyobraź sobie, że używałbyś asynchronicznej komunikacji w grze wieloosobowej. To byłaby katastrofa. Konieczność otwierania i zamykania połączenia za każdym razem, gdy jeden z graczy wysyła wiadomość do innego, spowodowałoby ogromne opóźnienia i opóźnienia między ruchami. W przypadku aplikacji dla wielu użytkowników w Przetwarzaniu, gdzie potrzebujemy komunikacji w czasie zbliżonym do rzeczywistego, używany jest inny typ połączenia, synchroniczny, nazywany połączeniem gniazdowym. Komunikacja synchroniczna jest również niezbędna w przypadku aplikacji na żywo, które wymagają interakcji między elementami w czasie rzeczywistym.



Asynchronous Request
(e.g., asking for a web page)



Socket Connection
(e.g., chat)

Połączenie z gniazdem sieciowym jest ciągłym połączeniem między dwoma programami w sieci. Gniazda są używane w aplikacjach dla wielu użytkowników, takich jak gry, komunikatory i czat (między innymi). Składają się z adresu IP - adresu numerycznego maszyny w sieci - oraz numeru portu - numeru służącego do kierowania informacji z jednego programu do drugiego. Możemy używać gniazd w przetwarzaniu do pisania programów, które mogą komunikować się ze sobą w czasie rzeczywistym. Aby to zrobić, użyjemy wbudowanej biblioteki net (processing.net) do stworzenia serwerów i klientów łączących się przez

gniazda.

19.2 Tworzenie serwera

Aby utworzyć serwer, musimy wybrać numer portu. Każdy klient, który chce połączyć się z serwerem, będzie musiał znać ten numer. Numery portów wahają się od 0 do 65 536, a każda liczba jest uczciwa, jednak porty od 0 do 1024 są zwykle zarezerwowane dla zwykłych usług, więc najlepiej ich unikać. Jeśli nie masz pewności, wyszukiwanie google wyszuka informacje o tym, czy port jest prawdopodobnie używany przez inną aplikację. Dla naszych celów użyjemy portu 5,204 (jest to ten sam port, który jest używany w odwołaniu do biblioteki Processing net, który można znaleźć na processing.org). Aby utworzyć serwer, musimy najpierw zaimportować bibliotekę i utworzyć instancję obiektu Server.

import processing.net. *;

Server servera;

Server jest inicjalizowany przez konstruktor, który pobiera dwa argumenty: „this” (odwołanie do tego apletu, jak wyjaśniono w rozdziale 16) oraz wartość całkowitą dla numeru portu.

server = new Server (this, 5204);

Serwer uruchamia się i czeka na połączenia, gdy tylko zostanie utworzone. Można go zamknąć w dowolnym momencie, wywołując funkcję stop ().

server.stop ();

Możesz przypomnieć sobie z naszej dyskusji o przechwytywaniu wideo w części 16, że użyliśmy funkcji wywołania zwrotnego (captureEvent ()) do obsługi nowej klatki wideo dostępnej z kamery. Możemy się dowiedzieć, czy nowy klient połączył się z naszym serwerem za pomocą tej samej techniki, używając funkcji callback serverEvent (). serverEvent () wymaga dwóch argumentów: serwera (generującego zdarzenie) i klienta (który się połączył). Możemy użyć tej funkcji, na przykład, aby pobrać adres IP podłączonego klienta.

// Funkcja serverEvent jest wywoływana za każdym razem

// Nowy klient się łączy

void serverEvent (server servera, client client) { //Zdarzenia serwera występują tylko wtedy, gdy
//łączy się nowy klient.

println ("Nowy klient połączył się:" + client.ip ());

}

Gdy klient wysyła wiadomość (po połączeniu), nie jest generowany serverEvent (). Zamiast tego musimy użyć funkcji available (), aby określić, czy jest dostępna nowa wiadomość od dowolnego klienta, którą można odczytać. Jeśli istnieje, zwracane jest odwołanie do transmisji rozgłoszeniowej klienta i możemy odczytać zawartość za pomocą metody readString (). Jeśli nic nie jest dostępne, funkcja zwróci wartość null, co oznacza brak wartości (lub brak obiektu klienta).

void draw() {

// Jeśli klient jest dostępny, dowiemy si

//Jeśli nie ma klienta, będzie on „null”

Client someClient = server.available();

// Powinniśmy kontynuować tylko wtedy, gdy klient nie jest pusty

if (someClient! = null) {

println("Client says: " + SomeClient.readString());

}

}

Funkcja readString() jest przydatna w aplikacjach, w których informacje tekstowe są przesyłane przez sieć. Jeśli dane powinny być traktowane inaczej, na przykład jako liczba (jak zobaczymy w przyszłych przykładach), można wywołać inne metody read(). Serwer może również wysyłać wiadomości do klientów, a to odbywa się za pomocą metody write(). server.write („Świetnie, dzięki za wiadomość! n”); W zależności od tego, co robisz, często dobrym pomysłem jest wysłanie znaku nowej linii na końcu wiadomości. Sekwencja ucieczki do dodawania znaku nowej linii do łańcucha to „n”. Łącząc wszystkie powyższe elementy, możemy napisać prosty serwer czatu. Ten serwer odpowiada na każdą otrzymaną wiadomość zwrotem „Jak to„ czujesz ”? ”Patrz przykład 19-1

Przykład 19-1: Serwer prostej terapii

// Importuj biblioteki sieciowe

```
import processing.net.*;
// Zadeklaruj serwer
Server server;
// Używane do wskazania nowej wiadomośc
float newMessageColor = 255;
PFont f;
String incomingMessage = " " ;
void setup() {
size(400,200);
//Utwórz serwer na porcie 5204
server = new Server(this, 5204);          //Ten szkic uruchamia serwer na porcie 5204.
f = createFont( " Arial " ,16,true);
}
void draw() {
background(newMessageColor);
// newMessageColor z czasem zmienia kolor na biały
newMessageColor = constrain(newMessageColor + 0.3,0,255);
textFont(f);
textAlign(CENTER);
fill(255);
text(incomingMessage,width/2,height/2); //Najnowsza wiadomość przychodząca jest wyświetlana w
                                        //oknie.
// Jeśli klient jest dostępny, dowiemy się
// Jeśli nie ma klienta, będzie on „null”
Client client = server.available();
// Powinniśmy kontynuować tylko wtedy, gdy klient nie jest pusty
if (client! = null) {
// Odbierz wiadomość                    //Wiadomość jest odczytywana za pomocą readString ().
                                        //Funkcja trim () służy do usunięcia dodatkowego podziału
                                        //linii, który pojawia się wraz z komunikatem
incomingMessage = client.readString();
incomingMessage = incomingMessage.trim();
```

```
// Okno komunikatu Drukuj Processing

println( " Client says: " + incomingMessage);

// Napisz wiadomość z powrotem (pamiętaj, że dotyczy to WSZYSTKICH klientów)

server.write( " How does " + incomingMessage + " make you feel?\n " );

//Zresetuj newMessageColor na kolor czarny     //Odpowiedź jest wysyłana za pomocą write ().

newMessageColor = 0;

}

}

//  Funkcja serverEvent jest wywoływana za każdym razem, gdy łączy się nowy klient..

void serverEvent(Server server, Client client) {

incomingMessage = " A new client has connected: " + client.ip();

println(incomingMessage);

// Zresetuj newMessageColor na kolor czarny

newMessageColor = 0;

}
```

Po uruchomieniu serwera możemy utworzyć klienta, który łączy się z serwerem. Ostatecznie przyjrzymy się przykładowi, w którym piszemy zarówno serwer, jak i klienta w przetwarzaniu. Aby jednak wykazać, że serwer faktycznie działa, możemy połączyć się z nim za pomocą dowolnej aplikacji klienckiej telnet. Telnet jest standardowym protokołem dla połączeń zdalnych, a wszystkie komputery zazwyczaj mają wbudowane możliwości telnetu. Na komputerze Mac uruchom terminal, w systemie Windows przejdź do wiersza polecenia. Polecam również korzystanie z PuTTY, darmowego klienta

telnet: http://www.chiark.greenend.org.uk/~sgtatham/putty/.

Ponieważ łączymy się z serwerem z tego samego komputera, na którym działa serwer, adres, pod który się telnetujemy, to localhost, co oznacza komputer lokalny, port 5,204. Możemy również użyć adresu 127.0.0.1. Jest to specjalny adres zarezerwowany dla programów na komputerze, aby rozmawiać ze sobą lokalnie (tzn. Na tej samej maszynie) i jest odpowiednikiem localhost. Gdybyśmy łączyli się z innego komputera, musielibyśmy znać adres IP komputera, na którym działa serwer. Klienci Telnet tradycyjnie wysyłają wiadomości do serwera, gdy użytkownik wpisuje dane. Powrót karetki i przesunięcie linii są zawarte w komunikacie i dlatego, gdy serwer odsyła odpowiedź, zauważysz, że „W jaki sposób przetwarzanie” i „sprawiają, że czujesz się” pojawiają się w osobnych wierszach.

Ćwiczenie 19-1: Używanie technik manipulacji łańcuchami z części 15, popraw Przykład 19-1, aby klient wysyłał znaki nowej linii, serwer usuwa je przed odpowiedzią na klienta. Będziesz chciał zmienić zmienną „incomingMessage”.

```
incomingMessage = client.readString ();

incomingMessage = przychodząca wiadomość. ______ (______, ______);
```

19.3 Tworzenie klienta

Po napisaniu serwera i przetestowaniu go za pomocą telnetu, możemy rozwinąć naszego klienta w Przetwarzaniu. Zaczynamy tak samo, jak w przypadku serwera, importując bibliotekę sieciową i deklarując wystąpienie obiektu klienta.

```
import processing.net.*;

Client client;
```

Konstruktor klienta wymaga trzech argumentów - „this", odwołując się ponownie do tego PApplet, adresu IP, z którym chcemy się połączyć (jako String), oraz numeru portu (jako liczby całkowitej).

```
client = new Client (this, „127.0.0.1", 5204);
```

Jeśli serwer działa na innym komputerze niż klient, musisz znać adres IP tego komputera serwera. Dodatkowo, jeśli nie ma serwera działającego na określonym IP i porcie, szkic Processing wyświetli komunikat o błędzie: „java.net.ConnectException: połączenie odrzucone", co oznacza, że serwer odrzucił klienta lub że nie ma serwera. Wysyłanie na serwer jest łatwe dzięki funkcji write ().

```
client.write („Hello!");
```

Odczytywanie wiadomości z serwera odbywa się za pomocą funkcji read (). Jednak metoda read () brzmi z serwera po jednym bajcie na raz. Aby odczytać całą wiadomość jako ciąg, używany jest readString (). Zanim będziemy mogli nawet zastanowić się nad odczytaniem z serwera, musimy być pewni, że jest coś do przeczytania. To sprawdzenie odbywa się za pomocą dostępnego (). available () zwraca liczbę bajtów, które czekają na odczyt. Możemy określić, czy coś czeka na odczyt, pytając, czy liczba bajtów jest większa niż zero.

```
if (client.available ()> 0) {

String message = client.readString ();

}
```

Korzystając z kodu z przykładu 18-1 (wejście z klawiatury), możemy stworzyć klienta przetwarzającego, który łączy się i komunikuje z naszym serwerem, wysyłając wiadomości wprowadzone przez użytkownika. Patrz przykład 19-2.

Przykład 19-2: Klient prostej terapii

```
// Importuj biblioteki sieciowe

import processing.net.*;

// Zadeklaruj klienta

Client client;

// Używane do wskazania nowej wiadomości

float newMessageColor = 0;

//  String do przechowywania tego, co mówi serwer

String messageFromServer = " " ;

//  String do przechowywania typów użytkownika

String typing = " " ;
```

```
PFont f;

void setup() {

size(400,200);

// Utwórz klient

client = new Client(this, " 127.0.0.1 " , 5204);   //Połącz się z serwerem pod adresem 127.0.0.1
                                                    //(localhost), port 5204

f = createFont( " Arial " ,16,true);

}

void draw() {

background(255);

// Wyświetl wiadomość z serwera

fill(newMessageColor);

textFont(f);

textAlign(CENTER);

text(messageFromServer,width/2,140);

// Zanik wiadomości z serwera na biały

newMessageColor = constrain(newMessageColor + 1,0,255);   //Nowa wiadomość zmienia kolor na
                                                          //biały, zwiększając jasność.

// Wyświetl instrukcje

fill(0);

text( "Type text and hit return to send to server. ",width/2,60);

//  Wyświetl tekst wpisany przez użytkownika

fill(0);

text(typing,width/2,80);

//Jeśli są dostępne informacje do przeczytania

if (client.available() > 0) {        //Wiemy, że jest wiadomość z serwera, gdy jest ona większa niż
                                     //dostępne zero bajtów.

// Przeczytaj to jako String

messageFromServer = client.readString();

// Ustaw jasność na 0

newMessageColor = 0;

}
```

```
}

// Proste wprowadzanie klawiatury użytkownika

void keyPressed() {

// Jeśli zostanie naciśnięty klawisz powrotu, zapisz ciąg i wyczyść go

if (key == '\n') {

// Gdy użytkownik kliknie Enter, zapisz zdanie na serwerze

client.write(typing);      //Gdy użytkownik kliknie Enter, ciąg znaków jest wysyłany do serwera.

typing = " " ;

} else {

typing = typing + key;

}

}
```

Ćwiczenie 19-2: Utwórz klienta i serwer, które ze sobą rozmawiają. Poproś klienta, aby wysyłał wiadomości wpisane przez użytkownika, a serwer odpowiadał autonomicznie. Na przykład można użyć technik analizowania ciągu znaków, aby odwrócić słowa wysłane przez klienta. Klient: „Jak się masz? ”Serwer:„ Jesteś jak? ”

19.4 Nadawanie

Teraz, gdy rozumiemy podstawy działania klientów i serwerów, możemy zbadać bardziej praktyczne zastosowania komunikacji sieciowej. W przykładach klient / serwer terapeuty potraktowaliśmy dane przesyłane przez sieć jako łańcuch, ale nie zawsze tak jest. W tej sekcji przyjrzymy się pisaniu serwera, który transmituje dane numeryczne do klientów. Jak to jest przydatne? Co zrobić, jeśli chcesz stale nadawać temperaturę na zewnątrz domu lub wycenę akcji lub ilość ruchu widzianego przez kamerę? Można skonfigurować pojedynczy komputer z uruchomionym serwerem przetwarzania do przetwarzania i transmisji tych informacji. Szkice klientów z dowolnego miejsca na świecie mogą łączyć się z tym komputerem, aby otrzymywać informacje. Aby zademonstrować strukturę takiego programu, napiszemy serwer, który nadaje liczbę z przedziału od 0 do 255 (możemy wysłać tylko jeden bajt na czas). Następnie spojrzymy na klientów, którzy pobierają dane i interpretują je na swój sposób. Oto serwer, który zwiększa liczbę losowo i nadaje ją.

Przykład 19-3: Serwer nadający numer (0–255)

```
// Importuj biblioteki sieciow

import processing.net.*;

// Zadeklaruj serwer

Server server;

PFont f;

int data = 0;

void setup() {
```

```
size(200,200);
// Utwórz serwer na porcie 5204
server = new Server(this, 5204);
f = createFont( " Arial " ,20,true);
}
void draw() {
background(255);
// Wyświetl dane
textFont(f);
textAlign(CENTER);
fill(0);
text(data,width/2,height/2);
//Rozsyłanie danych
server.write(data);
// Arbitralnie zmieniając wartość danych losowo
data = (data + int(random( – 2,4))) % 256;
}
// Funkcja serverEvent jest wywoływana za każdym razem, gdy łączy się nowy klient.
void serverEvent(Server server, Client client) {
println( " A new client has connected: " + client.ip());
}
```

Następnie napiszemy klienta, który otrzyma numer z serwera i użyje go do wypełnienia zmiennej. Przykład jest napisany przy założeniu, że serwer i klient działają na tym samym komputerze (możesz otworzyć oba przykłady i uruchomić je razem w Przetwarzaniu), ale w scenariuszu ze świata rzeczywistego prawdopodobnie tak nie będzie. Jeśli zdecydujesz się uruchomić serwer i klientów na różnych komputerach, komputery muszą być połączone w sieć lokalnie (za pośrednictwem routera lub koncentratora, sieci Ethernet lub Wi-Fi) lub w Internecie. Adres IP można znaleźć w ustawieniach sieciowych urządzenia.

Przykład 19-4: Wartości odczytu klienta jako kolor tła

```
// Importuj biblioteki sieciowe
import processing.net.*;
// Zadeklaruj klienta
Client client;
```

```
// Dane, które będziemy czytać z serwera
int data;
void setup() {
size(200,200);
// Utwórz klienta
client = new Client(this, "127.0.0.1", 5204);
}
void draw() {
if (client.available() > 0) {
data = client.read(); // Read data
}
background(data);         //Przychodzące dane służą do pokolorowania tła
}
// Importuj biblioteki sieciowe
import processing.net.*;
// Zadeklaruj klienta
Client client;
// Dane, które będziemy czytać z serwera
int data;
void setup() {
size(200,200);
smooth();
// Utwórz klienta
client = new Client(this, " 127.0.0.1 " , 5204);
}
void draw() {
if (client.available() > 0) {
data = client.read(); // Read data
}
background(255);
stroke(0);
```

```
fill(175);

translate(width/2,height/2);

float theta = (data/255.0) *

rotate(theta);

rectMode(CENTER);      //Przychodzące dane są używane do obracania kwadratu.

rect(0,0,64,64);

}
```

Ćwiczenie 19-3: Napisz klienta, który używa numeru nadawanego z serwera do sterowania położeniem kształtu.

19.5 Komunikacja z wieloma użytkownikami, część 1: Serwer

Przykład transmisji pokazuje komunikację jednokierunkową, w której serwer rozgłasza wiadomość i wielu klientów odbiera tę wiadomość. Jednak model transmisji nie pozwala klientowi odwrócić się i wysłać odpowiedzi z powrotem na serwer. W tej części omówimy sposób tworzenia szkicu, który obejmuje komunikację między wieloma klientami obsługiwanymi przez serwer. Zobaczmy, jak działa czat. Pięciu klientów (ty i czterech przyjaciół) łączy się z serwerem. Jeden klient wpisuje wiadomość: „Hej wszystkim! „Wiadomość w wiadomości jest wysyłana do serwera, który przekazuje ją z powrotem wszystkim pięciu klientom. Większość aplikacji dla wielu użytkowników działa w podobny sposób. Na przykład gra online dla wielu graczy prawdopodobnie wysyłałaby informacje związane z miejscem pobytu i działaniem na serwer, który transmituje te dane z powrotem do wszystkich innych klientów grających w grę. W aplikacji Przetwarzanie za pomocą biblioteki sieci można opracować aplikację dla wielu użytkowników. Aby zademonstrować, utworzymy wspólną tablicę sieciową. Gdy klient przeciąga myszą po ekranie, szkic wysyła współrzędne X, Y do serwera, który przekazuje je z powrotem do innych podłączonych klientów. Wszyscy połączeni będą mogli zobaczyć rysunki wszystkich innych. Oprócz uczenia się, jak komunikować się między wieloma klientami, w tym przykładzie omówiono sposób wysyłania wielu wartości. W jaki sposób klient może wysłać dwie wartości (współrzędną X i Y) i poinformować serwer, który z nich jest taki? Pierwszym krokiem w kierunku rozwiązania jest opracowanie protokołu komunikacji między klientami. W jakim formacie przesyłane są informacje i jak te informacje są otrzymywane i interpretowane? Na szczęście dla

my, czas spędzony na nauce tworzenia, zarządzania i analizowania obiektów String w częściach 17 i 18 zapewni wszystkie potrzebne narzędzia. Załóżmy, że klient chce wysłać lokalizację myszy: mouseX = 150 i mouseY = 125. Musimy sformatować tę informację jako ciąg w sposób dogodny do rozszyfrowania. Jedna z możliwości jest następująca:

„Pierwszą liczbą przed przecinkiem jest lokalizacja X, druga liczba po przecinku to 125. Nasze dane kończą się, gdy pojawia się gwiazdka (*). ”W kodzie wyglądałoby to następująco:

```
String dataToSend = "100,125 *";
```

lub, bardziej ogólnie:

```
String dataToSend = mouseX + "," + mouseY + "*";
```

Tutaj opracowaliśmy protokół wysyłania i odbierania danych. Wartości całkowite dla mouseX i mouseY są kodowane jako łańcuch podczas wysyłania (liczba, po której następuje przecinek, po której następuje numer, a po nim gwiazdka). Będzie musiał zostać zdekodowany po otrzymaniu, a do tego

dojdziemy później. Powinienem również zaznaczyć, że większość przykładów zazwyczaj używa znaku nowej linii lub powrotu karetki do oznaczenia końca wiadomości (jak widzieliśmy w pierwszej części tego rozdziału). Używamy tu gwiazdki z dwóch powodów: (1) gwiazdka jest wyraźnie widoczna, gdy jest wyświetlana, podczas gdy nowa linia nie jest (szczególnie w kontekście książki), a (2) za pomocą gwiazdki pokazuje, że można zaprojektować i wdrożyć dowolny protokół komunikacyjny wybierz tak długo, jak pasuje do kodu klienta i serwera.

Co jest naprawdę wysyłane?

Dane przesyłane przez sieć są wysyłane jako sekwencyjna lista pojedynczych bajtów. Przypominając omówienie typów danych w Part4, bajt jest liczbą 8-bitową, to znaczy liczbą składającą się z ośmiu 0 i 1 lub wartością z przedziału od 0 do 255.

Załóżmy, że chcemy wysłać liczbę 42. Mamy dwie opcje:

client.write (42); // wysłanie bajtu 42

W powyższym wierszu wysyłamy rzeczywisty bajt 42.

client.write („42”); // wysłanie ciągu „42”

W powyższym wierszu wysyłamy ciąg znaków. Th na String składa się z dwóch znaków, „4” i „2”. Wysyłamy dwa bajty! Te bajty są określane za pomocą kodu ASCII (American Standard Code for Information Interchange), znormalizowanego sposobu kodowania znaków. Znak „A” to bajt 65, znak „B” 66 i tak dalej. Znak „4” to bajt 52, a „2” to 50. Gdy czytamy dane, to od nas zależy, czy chcemy interpretować bajty wchodzące jako dosłowne wartości liczbowe czy jako kody ASCII dla znaków. Osiągamy to, wybierając odpowiednią funkcję read ().

int val = client.read (); // pasuje do client.write (42);

String s = client.readString (); // pasuje do client.write („42”);

int num = int (s); // przekonwertuj ciąg, który jest odczytywany na liczbę

Jesteśmy teraz gotowi do utworzenia serwera do odbierania wiadomości od klienta. Sformatowanie tych wiadomości za pomocą naszego protokołu będzie zadaniem klienta. Zadanie serwera pozostaje proste: (1) odbieranie danych i (2) przekazywanie danych. Jest to podobne do podejścia, które zastosowaliśmy w sekcji 19.2.

Krok 1 . Odbieranie danych.

```
Client client = server.available();

if (client ! = null) {

incomingMessage = client. readStringUntil( ' * ' );     // Ponieważ zaprojektowaliśmy nasz własny
                                                        //protokół i nie używamy znaku powrotu
                                                        //nowej linii / karetki do oznaczenia końca
                                                        //naszej wiadomości, musimy użyć
                                                        //readStringUntil () zamiast readString ().

}
```

Nowością w tym przykładzie jest funkcja readStringUntil (). Funkcja readStringUntil () przyjmuje jeden argument, znak. Ten znak jest używany do oznaczania końca przychodzących danych. Po prostu

postępujemy zgodnie z protokołem ustalonym podczas wysyłania. Jesteśmy w stanie to zrobić, ponieważ projektujemy zarówno serwer, jak i klienta. Po odczytaniu danych jesteśmy gotowi dodać:

Krok 2. Przekazywanie danych klientom.

```
Client client = server.available();

if (client ! = null) {

incomingMessage = client.readStringUntil( ' * ' );

server.write(incomingMessage);          // Zapisywanie wiadomości z powrotem do wszystkich
                                        //klientów.
```

Oto pełny serwer z dzwonkami i gwizdkami. Komunikat jest wyświetlany na ekranie, gdy łączą się nowi klienci, a także gdy serwer odbiera dane.

Przykład 19-6: Serwer dla wielu użytkowników

```
// Importuj biblioteki sieciow

import processing.net.*;

// Zadeklaruj serwer

Server server;

PFont f;

String incomingMessage = " " ;

void setup() {

size(400,200);

// Utwórz serwer na porcie 5204

server = new Server(this, 5204);

f = createFont( "Arial",20,true);

}

void draw() {

background(255);

// Wyświetl prostokąt z nowym kolorem wiadomości

fill(0);

textFont(f);

textAlign(CENTER);

text(incomingMessage,width/2,height/2);

// Jeśli klient jest dostępny, dowiemy się

// Jeśli nie ma klienta, będzie on „null”
```

```
Client client = server.available();

// Powinniśmy kontynuować tylko wtedy, gdy klient nie jest pusty

if (client ! = null) {

// Odbierz wiadomość

incomingMessage = client.readStringUntil( '*'); // Wszystkie wiadomości otrzymane od jednego
                                                //klienta są natychmiast przekazywane z powrotem do
                                                //wszystkich klientów za pomocą write()

// Okno komunikatu Print Processing

println( "Client says: " + incomingMessage);

// Napisz wiadomość z powrotem (pamiętaj, że dotyczy to WSZYSTKICH klientów)

server.write(incomingMessage);

}

}

// Funkcja serverEvent jest wywoływana za każdym razem, gdy łączy się nowy klient.

void serverEvent(Server server, Client client) {

incomingMessage = " A new client has connected: " + client.ip();

println(incomingMessage);

}
```

19.6 Komunikacja z wieloma użytkownikami, część 2: Klient

Zadanie klienta jest trojakie:

1. Wyślij współrzędne mouseX i mouseY na serwer.

2. Pobierz wiadomości z serwera.

3. Wyświetl elipsy w oknie na podstawie komunikatów serwera.

W przypadku kroku 1 musimy przestrzegać protokołu, który ustanowiliśmy w celu wysłania:

mouseX przecinek mouseY gwiazdka

```
String out = mouseX + "," + mouseY + "*";

client.write (out);
```

Pytanie pozostaje: kiedy jest odpowiedni czas na przesłanie tych informacji? Możemy wybrać wstawienie tych dwóch linii kodu do głównej pętli draw (), wysyłając współrzędne myszy w każdej klatce. Jednak w przypadku klienta tablicy wystarczy wysłać współrzędne, gdy użytkownik przeciągnie myszą po oknie.

Funkcja mouseDragged () jest funkcją obsługi zdarzeń podobną do funkcji mousePressed (). Zamiast wywoływać, gdy użytkownik kliknie myszą, jest wywoływany za każdym razem, gdy pojawia się

zdarzenie przeciągania, to znaczy naciśnięcie przycisku myszy i poruszanie myszą. Uwaga: funkcja jest wywoływana w sposób ciągły, gdy użytkownik przeciąga myszą. To tutaj wybieramy wysyłanie.

```
void mouseDragged () {

String out = mouseX + "," + mouseY + "*";

// Wyślij ciąg do serwera

client.write (out);

// Wydrukuj komunikat wskazujący, że wysłaliśmy dane

println ("Wysyłanie:" + out);

}
```

Krok 2, pobieranie wiadomości z serwera, działa podobnie jak klient terapii i przykłady klientów transmisji. Jedyną różnicą jest użycie readStringUntil (), które jest zgodne z protokołem „numer-komenda-gwiazdka”.

```
if (client.available ()> 0) {

// Odczytaj wiadomość jako ciąg znaków, wszystkie wiadomości kończą się gwiazdką

String in = client.readStringUntil ('*');

// Otrzymano komunikat drukowania

println ("Odbieranie:" + in);

}
```

Po umieszczeniu danych w obiekcie String można go interpretować za pomocą technik parsowania z Części 18. Po pierwsze, Łańcuch jest dzielony na tablicę Ciągów za pomocą przecinka (lub gwiazdki) jako ogranicznika.

```
String [] splitUp = split (in, ", *");
```

Tablica String jest następnie konwertowana na tablicę liczb całkowitych (długość: 2).

```
int [] vals = int (splitUp);
```

A te liczby całkowite są używane do wyświetlania elipsy.

```
wypełnić (255,100);

noStroke ();

elipsa (vals [0], vals [1], 16,16);
```

Oto cały szkic klienta:

Przykład 19-7: Klient dla tablicy wielu użytkowników

```
// Importuj biblioteki sieciowe

import processing.net.*;

// Zadeklaruj klienta
```

```
Client client;
void setup() {
size(200,200);
// Utwórz klienta
client = new Client(this, "127.0.0.1", 5204);
background(255);
smooth();
}
void draw() {
// Jeśli są dostępne informacje do odczytu z serwera
if (client.available() > 0) {
// Odczytaj wiadomość jako ciąg znaków, wszystkie wiadomości
String in = client.readStringUntil( '*');
// Otrzymano komunikat drukowania
println( "Receiving: " + in);
// Podziel String na tablicę liczb całkowitych
int[] vals = int(splitTokens(in, ",*"));     // Klient odczytuje wiadomości z serwera i analizuje je za
                                             //pomocą splitTokens () zgodnie z naszym protokołem.
// Renderuj elipsę na podstawie tych wartości
fill(0,100);
noStroke();
ellipse(vals[0],vals[1],16,16);
}
}
// Wysyłanie danych za każdym razem, gdy użytkownik przeciągnie myszę
void mouseDragged() {
// Umieść String wraz z naszym protokołem: mouseX comma mouseY gwiazdka
String out = mouseX + " ," + mouseY + " *";
// Wyślij String do serwera
client.write(out);                  // Wiadomość jest wysyłana za każdym razem, gdy mysz jest
                                    //przeciągnięta. Pamiętaj, że klient otrzyma własne wiadomości! Nic
                                    //nie jest tu wyciągnięte!
```

```
// Wyświetl komunikat wskazujący, że wysłaliśmy dan

println("Sending: " + out);

}
```

19.7 Komunikacja z wieloma użytkownikami, część 3: Wszystko razem teraz

Podczas uruchamiania aplikacji dla wielu użytkowników ważna jest kolejność uruchamiania elementów. Szkice klienta nie powiodą się, jeśli szkic serwera nie jest już uruchomiony. Najpierw należy (a) zidentyfikować adres IP serwera, (b) wybrać port i dodać go do kodu serwera, oraz (c) uruchomić serwer. Następnie możesz uruchomić klientów z poprawnym adresem IP i portem. Jeśli pracujesz nad projektem dla wielu użytkowników, najprawdopodobniej chcesz uruchomić serwery i klientów na osobnych komputerach. W końcu jest to cel tworzenia aplikacji dla wielu użytkowników. Jednak do celów testowania i rozwoju często wygodnie jest uruchamiać wszystkie elementy z jednego komputera. W tym przypadku adresem IP serwera będzie „localhost" lub 127.0.0.1 (należy zwrócić uwagę, że jest to adres IP użyty w przykładach tej części). Jak zostanie to omówione w sekcji 21.3, funkcja „Eksportuj do aplikacji" przetwarzania pozwoli Ci wyeksportować samodzielną aplikację na serwer, którą możesz następnie uruchomić w tle podczas rozwijania klienta w Przetwarzaniu. Szczegółowe informacje na temat działania funkcji „eksportuj do aplikacji" można znaleźć w części 18. Można również uruchomić wiele kopii autonomicznej aplikacji, aby symulować środowisko z więcej niż jednym klientem.

Ćwiczenie 19-4: Rozwiń tablicę, aby umożliwić kolor. Każdy klient powinien wysłać wartość czerwoną, zieloną i niebieską oprócz lokalizacji XY. Nie musisz wprowadzać żadnych zmian na serwerze, aby to działało.

Ćwiczenie 19-5: Stwórz grę Ponga dla dwóch graczy graną przez sieć. To jest złożone zadanie, więc buduj je powoli. Na przykład powinieneś najpierw uruchomić Ponga bez sieci. Trzeba wprowadzić zmiany na serwerze; w szczególności serwer będzie musiał przypisać graczom wiosło, które łączą (w lewo lub w prawo).

19.8 Komunikacja szeregowa

Dobrą nagrodą za poznanie tajników komunikacji sieciowej jest to, że komunikacja szeregowa w przetwarzaniu jest bardzo prosta. Komunikacja szeregowa polega na odczytywaniu bajtów z portu szeregowego komputera. Te bajty mogą pochodzić z zakupionego sprzętu (na przykład joysticka szeregowego) lub takiego, który sam projektujesz, budując obwód i programując mikrokontroler. Jeśli jednak chcesz dowiedzieć się więcej na temat fizycznego komputera. Wiring i Arduino to dwie fizyczne platformy obliczeniowe o otwartym kodzie opracowane w Instytucie Projektowania Interakcji Ivrea z językiem programowania wzorowanym na Processing. Komunikacja szeregowa odnosi się do procesu wysyłania danych po kolei, po jednym bajcie na raz. Oto jak dane zostały wysłane przez sieć w naszych przykładach klient / serwer. Biblioteka szeregowa przetwarzania jest przeznaczona do komunikacji szeregowej z komputerem z lokalnego urządzenia, najprawdopodobniej przez port USB (Universal Serial Bus). Termin „serial" odnosi się do portu szeregowego, zaprojektowanego do współpracy z modemami, który rzadko jest spotykany na nowszych komputerach.

Proces odczytu danych z portu szeregowego jest praktycznie identyczny z procesem znalezionym w przykładach sieciowych klient / serwer, z kilkoma wyjątkami. Po pierwsze, zamiast importować bibliotekę sieciową, importujemy bibliotekę szeregową i tworzymy obiekt Serial.

```
import processing.serial. *;
```

Serial port = new Serial (this, „COM1”, 9600);

Konstruktor szeregowy pobiera trzy argumenty. Pierwszy to zawsze „to”, odnosząc się do tego apletu. Argument 2 jest łańcuchem reprezentującym używany port komunikacyjny. Komputery oznaczają porty nazwą. Na komputerze prawdopodobnie będą to „COM1”, „COM2”, „COM3” i tak dalej. Na komputerach z systemem UNIX (takich jak MAC OS X) będą one oznaczone jako „/dev/tty.something”, gdzie „coś” reprezentuje urządzenie końcowe. Jeśli używasz urządzenia USB, prawdopodobnie będziesz musiał zainstalować sterowniki USB, zanim będzie dostępny port roboczy. Instrukcje, jak to zrobić, pracując z Arduino można znaleźć w przewodniku Arduino: http://www.arduino.cc/en/Guide/HomePage. Możesz także wydrukować listę dostępnych portów za pomocą funkcji list () biblioteki szeregowej, która zwraca tablicę obiektów String.

String [] portList = Serial.list ();

println (portList);

Jeśli port, którego chcesz użyć, jest pierwszym na liście, na przykład twoje wywołanie do konstruktora będzie wyglądać następująco:

String [] portList = Serial.list ();

Serial port = new Serial (this, portList [0], 9600);

Trzeci argument to szybkość, z jaką dane są przesyłane szeregowo, zazwyczaj 9600 bodów. Bajty są wysyłane przez port szeregowy za pomocą funkcji write (). Można przesyłać następujące typy danych: bajt, char, int, tablica bajtów i String.

Pamiętaj, że jeśli wysyłasz ciąg znaków, przesyłane dane są surowymi wartościami bajtów ASCII każdego znaku.

port.write (65); // Wysyłanie bajtu 65

Dane można odczytać za pomocą tych samych funkcji, które można znaleźć w klientach i serwerach: read (), readString () i readStringUntil (). Funkcja callback, serialEvent (), jest wyzwalana za każdym razem, gdy ma miejsce zdarzenie szeregowe, to znaczy, gdy są dostępne dane do odczytania.

void serialEvent (port szeregowy) {

int input = port.read ();

println („Surowe wejście:” + wejście);

}

Funkcja read () zwróci -1, jeśli nie ma nic do odczytu, zakładając jednak, że piszesz kod wewnątrz serialEvent (), zawsze będą dostępne dane. Poniżej znajduje się przykład, który odczytuje dane z portu szeregowego i używa go do pokolorowania tła szkicu.

Przykład 19-8: Odczyt z portu szeregowego

import processing.serial.*;

int val = 0; // Aby przechowywać dane z portu szeregowego, używane do pokolorowania tład

Serial port; // Obiekt portu szeregowegot

void setup() {

```
size(200,200);
// W przypadku, gdy chcesz zobaczyć listę dostępnych portów
// println(Serial.list());
// Używanie pierwszego dostępnego portu (może być inny na twoim komputerze)
port = new Serial(this, Serial.list()[0], 9600);    //Inicjowanie obiektu Serial za pomocą pierwszego
                                                    //portu na liście.
}
void draw() {
// Ustaw tło
background(val);        // Dane szeregowe służą do pokolorowania tła.
}
// Wywoływany, gdy jest coś do przeczytania
void serialEvent(Serial port) {
// Odczytaj dane
val = port.read();      // Dane z portu szeregowego są odczytywane w serialEvent () za pomocą
                        //funkcji read () i przypisywane do globalnej zmiennej „val”
// Do debugowania
// println( " Raw Input: " + input);
}
```

Dla odniesienia, jeśli używasz Arduino, oto odpowiedni kod:

```
int val;
void setup() {
beginSerial(9600);      // To nie jest kod przetwarzania! To jest kod Arduino. Więcej informacji na
                        //temat Arduino można znaleźć na stronie: http://www.arduino.cc/.
pinMode(3, INPUT);
}
void loop() {
val = analogRead(0);
Serial.print(val,BYTE);
}
```

19.9 Komunikacja szeregowa z uzgadnianiem

Często korzystne jest dodanie komponentu uzgadniania do kodu komunikacji szeregowej. Jeśli urządzenie sprzętowe wysyła bajty szybciej niż szkic przetwarzania może na przykład odczytywać informacje, może to spowodować opóźnienie szkicu. Wartości czujników mogą się spóźniać, powodując, że interakcja staje się myląca lub myląca dla użytkownika. Proces wysyłania informacji tylko na żądanie, zwany „handshaking", łagodzi to opóźnienie. Po uruchomieniu szkicu wyśle bajt na urządzenie sprzętowe z prośbą o dane.

Przykład 19-9: Uzgadnianie

```
void setup() {

size(200,200);

// W przypadku, gdy chcesz zobaczyć listę dostępnych portów

// println(Serial.list());

// Używanie pierwszego dostępnego portu (może być inny na twoim komputerze)

port = new Serial(this, Serial.list()[0], 9600);

// Żądaj wartości z urządzenia sprzętowego

port.write(65); // Bajt 65 mówi urządzeniu szeregowemu, że chcemy odbierać dane.

}
```

Po zakończeniu szkicowania bajtu wewnątrz serialEvent (), ponownie prosi o nową wartość.

```
// Wywoływany, gdy jest coś do przeczytania

void serialEvent(Serial port) {

// Odczytaj dane

val = port.read();.

// Do debugowaniu

// println( "Raw Input: " +

// Poproś o nową wartość

port.write(65);  // Po otrzymaniu bajtu odpowiadamy na następny

}
```

Dopóki urządzenie sprzętowe jest zaprojektowane do wysyłania wartości czujnika tylko na żądanie, wszelkie możliwe opóźnienia zostaną wyeliminowane. Oto poprawiony kod Arduino. Ten przykład nie obchodzi, jaki jest bajt żądania, tylko, że istnieje żądanie bajtu. Bardziej zaawansowana wersja może mieć inne odpowiedzi dla różnych żądań

```
int val;

void setup() {

beginSerial(9600);

pinMode(3, INPUT);
```

```
}

void loop() {

// Wysyłaj tylko, jeśli coś się pojawiło

if (Serial.available() > 0) {        // To nie jest kod przetwarzania! To jest kod Arduino. Więcej informacji
                                     // na temat Arduino można znaleźć na stronie:
                                     //http://www.arduino.cc/

Serial.read();

val = analogRead(0);

Serial.print(val,BYTE);

}

}
```

19.10 Komunikacja szeregowa z ciągami

W przypadkach, gdy trzeba pobrać wiele wartości z portu szeregowego (lub liczb większych niż 255), funkcja readStringUntil () jest przydatna. Załóżmy na przykład, że chcesz czytać z trzech czujników, używając wartości czerwonego, zielonego i niebieskiego komponentu koloru tła szkicu. Tutaj użyjemy tego samego protokołu zaprojektowanego w przykładzie tablicy dla wielu użytkowników. Poprosimy urządzenie sprzętowe (gdzie czujniki na żywo) o przesłanie danych w następujący sposób:

Sensor Value 1 COMMA Sensor Value 2 COMMA Sensor Value 3 ASTERISK

Na przykład:

104,5,76 *

Przykład 19-10: Komunikacja szeregowa z ciągami

```
import processing.serial.*;

int r,g,b; // Używane do kolorowania tła

Serial port; // Obiekt portu szeregowego

void setup() {

size(200,200);

// W przypadku, gdy chcesz zobaczyć listę dostępnych portów

// println(Serial.list());

//  Używanie pierwszego dostępnego portu (może być inny na twoim komputerze)

port = new Serial(this, Serial.list()[0], 9600);

// Poproś o wartości od razu

port.write(65);

}
```

```
void draw() {
// Ustaw tło
background(r,g,b);
}
// Wywoływany, gdy jest coś do przeczytania
void serialEvent(Serial port) {
// Odczytaj dane
String input = port.readStringUntil( ' * ' );      //Dane z portu szeregowego są odczytywane w
                                                   //serialEvent () przy użyciu funkcji readStringUntil () z
                                                   //„*” jako znakiem końca.
if (input ! = null) {
// Otrzymano komunikat drukowania
println( " Receiving: " + input);
//  Podziel String na tablicę liczb całkowitych
int[] vals = int(splitTokens(input, " ,* " ));     //Dane są dzielone na tablicę ciągów z przecinkiem lub
                                                   //gwiazdką jako separator i konwertowane na tablicę
                                                   //liczb całkowitych
// Wypełnij zmienne r, g, b
r = vals[0];
g = vals[1];                //Trzy zmienne globalne to tablica wypełniona danymi wejściowymi
b = vals[2];
}
// Po zakończeniu ponownie poproś o wartości
port.write(65);
}
```

Odpowiedni kod Arduino:

```
int sensor1 = 0;
int sensor2 = 0;
int sensor3 = 0;
void setup()
{
beginSerial(9600);
```

```
pinMode(3, INPUT);
}
void loop()
{
if (Serial.available() > 0) { //wysyłaj tylko, jeśli usłyszałeś z powrote
Serial.read();
sensor1 = analogRead(0);
sensor2 = analogRead(1);
sensor3 = analogRead(2);
// Wyślij liczbę całkowitą jako łańcuch używając „DEC
Serial.print(sensor1,DEC);
// Wyślij przecinek - kod ASCII 44
Serial.print( ",", BYTE);
Serial.print(sensor2,DEC);
Serial.print( ",", BYTE);
Serial.print(sensor3,DEC);
//Wyślij gwiazdkę - kod ASCII 42
Serial.print( "*", BYTE);
}
}
```

Ćwiczenie 19-6: Jeśli masz kartę Arduino, zbuduj swój własny interfejs, aby kontrolować już wykonany szkic przetwarzania. (Zanim podejmiesz taką próbę, powinieneś upewnić się, że możesz pomyślnie uruchomić proste przykłady podane w tej części).

Projekt Lekcji Ósmej

Utwórz wizualizację danych, ładując informacje zewnętrzne (plik lokalny, strona internetowa, kanał XML, serwer lub połączenie szeregowe) do przetwarzania. Upewnij się, że budujesz projekt stopniowo. Na przykład spróbuj zaprojektować wizualizację najpierw bez żadnych rzeczywistych danych (użyj liczb losowych lub wartości zakodowanych na stałe). Jeśli ładujesz dane z sieci, rozważ użycie lokalnego pliku podczas tworzenia projektu. Nie bój się fałszować danych, czekając, aż skończysz z aspektami projektu przed połączeniem rzeczywistych danych. Eksperymentuj z poziomami abstrakcji. Spróbuj wyświetlić informacje dosłownie na ekranie, pisząc tekst. Zbuduj abstrakcyjny system, w którym dane wejściowe wpływają na zachowania obiektów (możesz nawet użyć „ekosystemu" z projektu lekcji szóstej). Użyj miejsca podanego poniżej, aby naszkicować projekty, notatki i pseudokod dla twojego projektu.

Dźwięk

Przetwarzanie nie ma wbudowanej obsługi dźwięku. Nie, że coś jest nie tak z dźwiękiem. Nie, że Processing ma osobistą urazę wobec dźwięku. Po prostu nie ma wbudowanego dźwięku. Jak omówiliśmy we wstępie, Processing jest językiem programowania i środowiskiem programistycznym, zakorzenionym w Javie, przeznaczonym do nauki programowania w kontekście wizualnym. Jeśli więc chcesz tworzyć wielkoskalowe aplikacje interaktywne skupione głównie na dźwięku, powinieneś naprawdę zadać sobie pytanie: „Czy Processing jest dla mnie odpowiednim środowiskiem programistycznym? ”Ta część pomoże odpowiedzieć na to pytanie, ponieważ badamy możliwości i ograniczenia pracy z dźwiękiem w przetwarzaniu. Włączenie dźwięku do szkiców przetwarzania można wykonać na wiele różnych sposobów. Ponieważ Processing nie obsługuje dźwięku w swojej podstawowej bibliotece, wielu programistów Processing decyduje się na włączenie dźwięku za pośrednictwem aplikacji innych firm, ukierunkowanej na dźwięk, takiej jak PureData (http://www.puredata.org/) lub Max / MSP (http: / /www.cycling74.com/). Przetwarzanie może komunikować się z tymi aplikacjami poprzez OSC („open sound control”) protokół komunikacji sieciowej między komputerami i urządzeniami multimedialnymi. Można to osiągnąć w przetwarzaniu za pomocą biblioteki sieciowej (patrz poprzedni rozdział) lub w bibliotece oscP5, autorstwa Andreasa Schlegla (http://www.sojamo.de/libraries/oscP5). lista udostępnionych bibliotek do używania dźwięku bezpośrednio w przetwarzaniu: odtwarzanie próbek dźwięku, analizowanie wejścia dźwięku z mikrofonu, syntezowanie dźwięku oraz wysyłanie i odbieranie informacji midi. Rozdział ten skupi się na dwóch z tych elementów: odtwarzaniu dźwięku i wprowadzaniu dźwięku. Do odtwarzania efektów dźwiękowych przyjrzymy się bibliotece Sonia (autorstwa Amit Pitaru) i bibliotece Minim (autorstwa Damien Di Fede). Sygnał dźwiękowy zostanie pokazany w Sonia. Te biblioteki dźwięków są dostępne pod następującymi adresami URL:

Sonia: http://sonia.pitaru.com/

Minim: http://code.compartmental.net/tools/minim/

Aby zapoznać się z przeglądem instalacji bibliotek innych firm, odwiedź część 12. Będziesz także chciał odwiedzić

http: //www.learning processing.com, aby pobrać przykładowe pliki dźwiękowe użyte w tych przykładach.

20.1 Naprawdę prosty dźwięk

Zanim jednak przejdziemy do bibliotek, istnieje jeden prosty (ale bardzo ograniczony) sposób odtwarzania pliku dźwiękowego w szkicu przetwarzania bez instalowania biblioteki innej firmy. Dzieje się tak dzięki użyciu klasy Movie, której nauczyliśmy się w części 16. Klasa filmu jest przeznaczona do odtwarzania filmów QuickTime, ale może być również używana do odtwarzania pliku WAV lub AIFF. WAV to format plików audio, który oznacza „format dźwięku Waveform” i jest standardowym formatem dla komputerów PC. AIFF oznacza „Audio Interchange File Format” i jest powszechnie używany na komputerach Macintosh. Wszystkie funkcje, które obejrzeliśmy w części 16, są dostępne dla pliku dźwiękowego. Główną różnicą jest oczywiście to, że nie możemy użyć image () do wyświetlenia obiektu Movie, ponieważ nie ma obrazu.

```
import processing.video.*;

Movie movie;

void setup() {
```

```
size(200, 200);

movie = new Movie (this, " dingdong.wav " );     //Utwórz obiekt filmu za pomocą pliku WAV (lub AIFF).

}

void draw() {

background(0);

noLoop();

}

void mousePressed() {

movie.stop();

movie.play();     //Dźwięk jest odtwarzany po naciśnięciu myszy. Włączenie funkcji stop() przed play()
                  //zapewnia odtwarzanie pliku dźwiękowego od początku.
```

Ćwiczenie 20-1: Korzystając ze szkicu odbijającej piłki z przykładu 5–6, zagraj efekt dźwiękowy za każdym razem, gdy piłka odbija się od krawędzi okna.

20.2 Rozpoczęcie pracy z Sonia i Minim

Klasa Movie będzie służyć do odtwarzania prostych efektów dźwiękowych, ale dla bardziej zaawansowanych funkcji odtwarzania będziemy musieli spojrzeć na biblioteki przeznaczone do użytku z dźwiękiem. Zaczniemy od Sonia (http://sonia.pitaru.com/). Po pierwsze, tak jak w przypadku każdej biblioteki, której potrzebujesz, najpierw potrzebujesz instrukcji importu na górze kodu.

```
importuj pitaru.sonia_v2_9. *;
```

Sonia to interfejs do biblioteki dźwiękowej Java, zatytułowanej Jsyn autorstwa Phila Burka. Aby działała, Sonia musi komunikować się z odtwarzanym dźwiękiem i sprzętem wejściowym komputera. Ta komunikacja odbywa się za pośrednictwem Jsyn. Dlatego przed odtworzeniem jakichkolwiek dźwięków należy ustanowić to łącze, czyli uruchomić silnik dźwięku. Robi się to za pomocą następującego wiersza kodu, który należy umieścić w setup ().

```
void setup () {

Sonia.start (to);

}
```

Teraz byłoby to nieodpowiedzialne z naszej strony, aby połączyć się z dźwiękowym sprzętem, ale nigdy nie zawracaj sobie głowę zamknięciem, kiedy skończymy. Do tej pory przepływ szkicu przetwarzania jest zbyt znajomy, zacznij od setup(), zapętl z draw() i zatrzymaj się za każdym razem, gdy wyjdziesz ze szkicu. Chodzi o to aby zakończyć silnik dźwięku podczas ostatniego kroku, gdy szkic się skończy. Na szczęście jest dodatkowy ukryta funkcja, którą możemy zaimplementować, zatytułowana stop (), która właśnie to robi. Kiedy szkic się kończy, jakikolwiek instrukcje last minute mogą być wykonywane wewnątrz funkcji stop (). To jest miejsce, w którym zamkniemy silnik Sonii.

```
void stop () {

Sonia.stop ();
```

```
super.stop ();

}
```

Sonia.stop () to wezwanie do wstrzymania Soni. Jest jednak również linia super.stop (). Znaczenie „super” zostanie omówione dalej w Części 22. Na razie możemy zrozumieć, że ten wiersz kodu mówi „Och, cokolwiek jeszcze normalnie zrobiłbyś w stop (), rób to też”. powyższe dotyczy również Minim. Oto odpowiedni kod:

```
import ddf.minim.*;

void setup() {

size(200, 200);

Minim.start(this);          //Uruchom silnik dźwięku Minim!

}

void stop() {      //Z Minim, dźwięki zostaną tutaj zamknięte indywidualnie. Zobaczymy to za chwilę

super.stop();

}

dingdong = new Sample( "dingdong.wav");  //Plik „dingdong.wav” należy umieścić w katalogu danych.
```

Podobnie jak w przypadku obrazów, ładowanie pliku dźwiękowego z dysku twardego jest procesem powolnym, więc poprzednia linia kodu powinna być umieszczona w setup (), aby nie przeszkadzać w szybkości rysowania (). Typ pliku dźwiękowego kompatybilnego z Sonia jest nieco ograniczony. Dozwolone są tylko dźwięki sformatowane jako pliki WAV lub AIFF (16-bitowe mono lub stereo). Jeśli chcesz użyć pliku dźwiękowego, który nie jest zapisany w kompatybilnym formacie, możesz pobrać darmowy edytor audio, taki jak Audacity (http://audacity.sourceforge.net/) i przekonwertować plik. Większość plików audio może być zapisana jako surowa lub z kompresją. Sonia działa tylko z plikami raw. Jednak w następnej sekcji pokażemy, w jaki sposób pliki skompresowane (MP3) można odtwarzać w bibliotece ESS. Po załadowaniu dźwięku odtwarzanie jest łatwe.

```
dingdong.play ();          //Funkcja play() odtwarza próbkę dźwięku raz
```

Poniższy przykład odtwarza dźwięk dzwonka przy każdym kliknięciu myszką na dzwonek graficzny. Ta klasa Doorbells realizuje prostą funkcjonalność przycisków (najazd i kliknięcie) i okazuje się być rozwiązaniem dla ćwiczenia 9–8.

Przykład 20-2: Dzwonek do drzwi z Sonia

```
// Import biblioteki Sonia

import pitaru.sonia_v2_9.*;

// Przykładowy obiekt (dla dźwięku))

Sample dingdong;

// Obiekt dzwonka (który wywoła dźwięk)

Doorbell doorbell;

void setup() {
```

```
size(200,200);
Sonia.start(this); // Start Sonia engine.
//  Utwórz nowy przykładowy obiekt
dingdong = new Sample( " dingdong.wav " );
// Utwórz nowy dzwonek do drzwi
doorbell = new Doorbell(150,100,32);
smooth();
}
void draw() {
background(255);
//Pokaż dzwonek do drzwi
doorbell.display(mouseX,mouseY);
}
void mousePressed() {
// Jeśli użytkownik kliknie dzwonek, odtwórz dźwięk!
if (doorbell.contains(mouseX,mouseY)) {
dingdong.play();
}
}
// Zamknij silnik dźwięku
public void stop() {
Sonia.stop();
super.stop();
}
// A Klasa opisująca „dzwonek do drzwi” (naprawdę przycisk)
class Doorbell {
//Lokalizacja i rozmiar
float x;
float y;
float r;
// Stwórz dzwonek do drzwi
```

```
Doorbell (float x_, float y_, float r_) {
x = x_;
y = y_;
r = r_;
}
// Czy jest punkt wewnątrz dzwonka (używany do przesuwania myszy itp.))
boolean contains(float mx, float my) {
if (dist(mx,my,x,y) < r) {
return true;
} else {
return false;
}
}
// Pokaż dzwonek (zakodowane kolory, można poprawić)
void display(float mx, float my) {
if (contains(mx,my)) {
fill(100);
} else {
fill(175);
}
stroke(0);
ellipse(x,y,r,r);
}
}
```

Ćwiczenie 20-2: Przepisz klasę Dzwonek, aby zamieścić odniesienie do próbki Sonia. Umożliwi to łatwe tworzenie wielu obiektów dzwonkowych, z których każdy będzie odtwarzał inny dźwięk. Oto kod, który pozwoli Ci zacząć.

```
// Klasa opisująca „dzwonek do drzwi” (naprawdę przycisk)
class Doorbell {
// Lokalizacja i rozmiar
float x;
float y;
```

```
float r;

//  Przykładowy obiekt (dla dźwięku)

_________ _____________;

// Stwórz dzwonek do drzwi

Doorbell (float x_, float y_, float r_, _______ filename) {

x = x_;

y = y_;

r = r_;

_________ = new _________(_____________);

}

void ring() {

_____________________;

}

boolean contains(float mx, float my) {

// taki sam jak oryginał

}

void display(float mx, float my) {

// taki sam jak oryginał

}

}
```

Jeśli uruchomisz przykład dzwonka i klikniesz dzwonek wielokrotnie w krótkich odstępach czasu, zauważysz, że dźwięk uruchamia się ponownie po każdym kliknięciu. Nie ma szansy na pierwsze zakończenie gry. Chociaż nie stanowi to większego problemu dla tego prostego przykładu, zatrzymanie dźwięku przed ponownym uruchomieniem może być bardzo ważne w innych, bardziej złożonych szkicach dźwiękowych. Najprostszym sposobem osiągnięcia takiego wyniku jest zawsze sprawdzenie i sprawdzenie, czy dźwięk jest odtwarzany przed wywołaniem funkcji play (). Sonia ma fajną funkcję o nazwie isPlaying (), która robi dokładnie to, zwracając true lub false. Chcemy odtworzyć dźwięk, jeśli jeszcze nie gra, to znaczy

```
if (! dingdong.isPlaying ()) {        //Pamiętaj, „!” Oznacza nie!

dingdong.play ();

}
```

Ćwiczenie 20-3: Rozwiń klasę Dzwonek do animacji (być może przesuwając swoją lokalizację i losowo zmieniając jej rozmiar) tylko podczas odtwarzania dźwięku. Utwórz tablicę pięciu obiektów dzwonkowych. Oto część funkcji, którą należy dodać do klasy dzwonka:

```
void jiggle() {

if (_____)) {

x += _____;

y += _____;

_____;

}

}
```

Z Minim zamiast Sonia, bardzo mało zmian. Zamiast obiektu Sample mamy obiekt AudioPlayer. Poniższy przykład zawiera rozwiązania dla ćwiczeń 20-2 i 20-3 (ale używa raczej kodu Minim niż Sonia). Jednak nie implementuje tablicy.

Przykład 20-3: Dzwonek do drzwi z Minim

```
import ddf.minim.*;

// Obiekt dzwonka (który wywoła dźwięk)

Doorbell doorbell;

void setup() {

size(200,oim Minim

Minim.start(this);

// Utwórz nowy dzwonek do drzwi

doorbell = new Doorbell(150,100,32, "dingdong.wav");

smooth();

}

void draw() {

background(100,100,126);

// Pokaż dzwonek do drzwi

doorbell.display(mouseX,mouseY);

doorbell.jiggle();

}

void mousePressed() {

//Jeśli użytkownik kliknie dzwonek, odtwórz dźwięk!

if (doorbell.contains(mouseX,mouseY)) {

doorbell.ring();

}
```

```
}
// Zamknij pliki dźwiękowy
public void stop() {
doorbell.close();          //Obiekt dzwonka musi zamknąć dźwięk
super.stop();
}
class Doorbell {
// Lokalizacja i rozmiar
float x;
float y;
float r;
// Obiekt AudioPlayer
AudioPlayer dingdong;          //Obiekt AudioPlayer jest używany do przechowywania dźwięku.
// Stwórz dzwonek do drzwi
Doorbell (float x_, float y_, float r_, String filename) {
x = x_;
y = y_;
r = r_;
// załaduj „dingdong.wav” do nowego AudioPlayera
dingdong = Minim.loadFile(filename);
}
//Jeśli „dzwonek do drzwi” dzwoni, kształt trzęsie się
void jiggle() {
if (dingdong.isPlaying()) {          //Dzwonek tylko trzęsie się, jeśli gra dźwięk.
x + = random( – 1,1);
y + = random( – 1,1);
r = constrain(r + random( – 2,2),10,100);
}
}
// Dzwonki do drzwi!
```

```
void ring() {            //Funkcja dzwonka () odtwarza dźwięk, o ile nie jest już odtwarzany. rewind ()
                         //zapewnia, że dźwięk zaczyna się od początku
if (!dingdong.isPlaying()) {
dingdong.rewind();
dingdong.play();
}
}
// Czy jest punkt wewnątrz dzwonka (używany do przesuwania myszy itp.)
boolean contains(float mx, float my) {
if (dist(mx,my,x,y) < r) {
return true;
} else {
return false;
}
}
// Pokaż dzwonek (zakodowane kolory, można poprawić
void display(float mx, float my) {
if (contains(mx,my)) {
fill( 126,114,100);
} else {
fill(119,152,202);
}
stroke(202,175,142);
ellipse(x,y,r,r);
}
void close() {
dingdong.close();        //Dzwonek ma funkcję close(), aby zamknąć obiekt AudioPlayer.
}
}
```

Jedną z zalet korzystania z Minim over Sonia jest to, że Minim obsługuje pliki MP3. Pliki MP3 (lub „MPEG-1 Audio Layer 3”) są skompresowane i dlatego zajmują znacznie mniej miejsca na dysku twardym niż surowe pliki WAV lub AIFF.

AudioPlayer dingdong = Minim.loadFile ("dingdong.mp3");

20.4 Trochę bardziej wyszukane odtwarzanie dźwięku

Podczas odtwarzania próbkę dźwięku można manipulować w czasie rzeczywistym. Głośność, wysokość dźwięku i patelnię można kontrolować za pomocą Sonia i Minim. Zacznijmy od dźwieku w Sonia. Objętość przykładowego obiektu można ustawić za pomocą funkcji setVolume(), która przyjmuje wartość punktu płynięcia między 0,0 a 1,0 (milczenie 0,0, najgłośniejsze 1,0). Poniższy fragment zakłada próbkę o nazwie „tone" i ustawia głośność na podstawie pozycji mouseX (normalizując ją do zakresu od 0 do 1).

float ratio= (float) mouseX / width; //Zakresy głośności od 0,0 do 1,0

tone.setVolume (ratio);

Pan działa w ten sam sposób, tylko zakres wynosi od -1,0 (dla lewej) i 1,0 (dla prawej).

float ratio = (float) mouseX / width;

tone.setPan (współczynnik * 2 - 1); //Pan waha się od –1,0 do 1,0.

Skok jest zmieniany przez zmianę szybkości odtwarzania (tj. Szybsze odtwarzanie jest wyższym skokiem, wolniejsze odtwarzanie jest niższym skokiem) przy użyciu setRate (). Możesz ustawić szybkość odtwarzania próbek na dowolne, dość obszerny zakres wynosi od 0 (gdzie w ogóle go nie słyszysz) do 88 200 (stosunkowo szybka szybkość odtwarzania).

float rate = (float) mouseX / width;

tone.setRate (stosunek * 88200); // Rozsądny zakres stawki (tj. Skoku) wynosi od 0 do 88 200

Poniższy przykład dostosowuje głośność i wysokość dźwięku zgodnie z ruchami myszy. Zwróć uwagę na użycie repeat() zamiast play(), które w kółko odtwarza dźwięk zamiast go odtwarzać.

Przykład 20-4: Manipulowanie dźwiękiem (z Sonia)

// Importuj bibliotekę Sonia

import pitaru.sonia_v2_9.*;

// Obiekt Sample (dla dźwięku)

Sample tone;

void setup() {

size(200,200);

Sonia.start(this); // Uruchom silnik Sonia.

// Utwórz nowy przykładowy obiekt.

tone = new Sample(" tone.wav ");

// Zapętl dźwięk na zawsz

// (cóż, przynajmniej dopóki nie zostanie wywołany stop()

tone.repeat();

```
smooth();
}
void draw() {
if (tone.isPlaying()) {
background(255);
} else {
background(100);
}
// Ustaw głośność w zakresie od 0 do 1,0
float ratio = (float) mouseX / width;                    // Głośność jest ustawiana zgodnie z pozycją mouseX
tone.setVolume(ratio);
// Ustaw stawkę na zakres od 0 do 88 20 0
// Zmiana zakresu zmienia ton
ratio = (float) mouseY / height;
tone.setRate(ratio*88200);                               // Szybkość jest ustawiana zgodnie z pozycją mouseY
// Narysuj kilka prostokątów, aby pokazać, co się dzieje
stroke(0);
fill(175);
rect(0,160,mouseX,20);
stroke(0);
fill(175);
rect(160,0,20,mouseY);
}
// Naciśnięcie myszy zatrzymuje się i uruchamia dźwięk
void mousePressed() {
if (tone.isPlaying()) {
tone.stop();                                 // Dźwięk można zatrzymać za pomocą funkcji stop()
} else {
tone.repeat();
}
}
```

// Zamknij silnik dźwięku

public void stop() {

Sonia.stop();

super.stop();

}

Ćwiczenie 20-4: W przykładzie 20-4, użyj osi Y, aby niższy dźwięk był odtwarzany, gdy mysz jest w dół, a nie w górę. Obiekt AudioPlayer można również manipulować za pomocą Minim przy użyciu funkcji: setVolume (), setPan () i addEff ect (), między innymi udokumentowanych na stronie Minim (http: //code.compartmental. Net / tools / minim /) .

20.5 Wejście na żywo

W części 16 przyjrzeliśmy się, w jaki sposób komunikacja szeregowa umożliwia szkicowi Processing odpowiedź na sygnał z zewnętrznego urządzenia sprzętowego podłączonego do czujnika. Odczyt wejścia z mikrofonu jest podobnym pościgiem. W istocie mikrofon działa jak czujnik. Mikrofon może nie tylko nagrywać dźwięk, ale może również określić, czy dźwięk jest głośny, cichy, wysoki, niski i tak dalej. Na przykład szkic Processing może określić, czy mieszka w zatłoczonym pokoju na podstawie poziomów dźwięku, czy też słucha sopranu lub basisty w oparciu o poziomy dźwięku. W tej sekcji opisano sposób pobierania i używania danych głośności za pomocą biblioteki Sonia. Aby przeanalizować poziomy dźwięku z mikrofonu, odwiedź stronę internetową Sonia (http://sonia.pitaru.com) w celu uzyskania dalszych przykładów. Poprzednie sekcje wykorzystywały obiekt Sample do odtwarzania dźwięku. Wejście dźwięku z mikrofonu jest pobierane za pomocą obiektu LiveInput. Istnieje trochę dziwne rozróżnienie w sposobie, w jaki wykorzystamy te dwie klasy, które musimy jeszcze spotkać w trakcie kursu. Rozważmy scenariusz, w którym mamy trzy pliki dźwiękowe. Tworzymy trzy przykładowe obiekty.

Próbka sample1 = nowa próbka („file1.wav”);

Próbka sample2 = nowa próbka („file2.wav”);

Próbka sample3 = nowa próbka („file3.wav”);

Technicznie rzecz biorąc, stworzyliśmy trzy przykłady obiektów przykładowych, urodzonych za pomocą klasy Sample. Jeśli my

chcemy odtworzyć dźwięk, musimy odwołać się do konkretnego obiektu przykładowego.

sample1.play ();

Dźwięk można zatrzymać, funkcją stop(). Jednak z LiveInput nie zamierzamy tworzyć żadnych obiektów. Ponieważ biblioteka Sonia zezwala tylko na jedno źródło wejściowe, 1 nie ma powodu, aby odwoływać się do konkretnego obiektu LiveInput. Zamiast tego, słuchając mikrofonu, odnosimy się do klasy LiveInput jako całości:

LiveInput.start (); // Uruchom LiveInput

Funkcje, które wywołujemy z samej nazwy klasy (a nie z konkretnej instancji obiektu), nazywane są funkcjami statycznymi. Nie możemy samodzielnie tworzyć funkcji statycznych w Przetwarzaniu, więc nie musimy się tym zbytnio przejmować. Jednakże, ponieważ są one częścią Java, napotkamy je od

czasu do czasu podczas korzystania z bibliotek współdzielonych. LiveInput to przykład klasy z funkcjami statycznymi do pobierania poziomów dźwięku i wysokości dźwięku z mikrofonu. Zacznijmy od zbudowania bardzo prostego przykładu, który wiąże rozmiar koła z poziomem dźwięku mikrofonu.

Krok 1 jest dokładnie tym, co właśnie zrobiliśmy, rozpocznij proces słuchania mikrofonu komputera.

LiveInput.start (); // Uruchom LiveInput

Krok 2 to odczyt poziomu głośności z samego mikrofonu. Możemy to zrobić na dwa sposoby. Ponieważ mikrofon słucha w stereo, możemy odczytać poziom głośności z prawego lub lewego kanału za pomocą getLevel ():

float rightLevel = LiveInput.getLevel (Sonia.RIGHT);

float leftLevel = LiveInput.getLevel (Sonia.LEFT);

Możemy również po prostu słuchać obu kanałów.

float level = LiveInput.getLevel ();

Zwrócony poziom zawsze będzie wynosił od 0,0 do 1,0. Łącząc to wszystko i wiążąc poziom dźwięku z wielkością elipsy, mamy:

Przykład 20-5: Wejście żywe z Sonia

```
// Importuj bibliotekę Sonia

import pitaru.sonia_v2_9.*;

void setup() {

size(200,200);

Sonia.start(this);

// Rozpocznij słuchanie mikrofon

LiveInput.start();      //Wszystkie funkcje wejścia dźwięku są statyczne, co oznacza, że są
                        //wywoływane z samej nazwy klasy, LiveInput, a nie z instancji obiektu.

smooth();

}

void draw() {

background(255,120,0);

// Pobierz ogólną głośność (od 0 do 1,0)

float level = LiveInput.getLevel();

fill(200);

stroke(50);

// Narysuj elipsę o rozmiarze opartym na głośności

ellipse(width/2,height/2,level*200,level*200);  // Zmienna przechowująca wolumin („poziom”) jest
                                                 //używana jako rozmiar elipsy.
```

```
}

// Zamknij silnik dźwięku

public void stop() {

Sonia.stop();

super.stop();

}
```

Ćwiczenie 20-5: Przepisz przykład 20-5 z lewymi i prawymi poziomami głośności mapowanymi do różnych okręgów

20.6 Próg dźwięku

Wspólna interakcja dźwiękowa uruchamia zdarzenie, gdy emitowany jest dźwięk. Rozważmy „klapy". „Klaskaj, zapalają się światła. Klasknij ponownie, światła zgasną. Klaskanie można uznać za bardzo głośny i krótki dźwięk. Aby zaprogramować „klapę" w Przetwarzaniu, będziemy musieli nasłuchiwać głośności i wywołać zdarzenie, gdy głośność jest wysoka. W przypadku klaskania możemy zdecydować, że gdy całkowita objętość jest większa niż 0,5, użytkownik klaskuje (nie jest to pomiar naukowy, ale jest wystarczający dla tego przykładu). Wartość która wynosi 0,5 jest znana jako próg. Powyżej progu wyzwalane są zdarzenia, poniżej ich nie ma.

```
float vol = LiveInput.getLevel ();

jeśli (vol> 0.5) {

// ZRÓB COŚ COŚ, GDYGŁOŚNOŚĆ JEST WIĘKSZA NIŻ JEDEN!

}
```

Przykład 20-6 rysuje prostokąty w oknie, gdy ogólny poziom głośności jest większy niż 0,5. Poziom głośności jest również wyświetlany po lewej stronie jako pasek.

Przykład 20-6: Próg dźwięku z Sonia

```
// Importuj bibliotekę Sonia

import pitaru.sonia_v2_9.*;

void setup() {

size(200,200);

Sonia.start(this); // Uruchom silnik Sonia.

LiveInput.start(); // Rozpocznij słuchanie mikrofonu

smooth();

background(255);

}

void draw() {

// Pobierz ogólną głośność (od 0 do 1,0))
```

```
float vol = LiveInput.getLevel();

// Jeśli głośność jest większa niż 0,5, narysuj prostokąt

if (vol > 0.5) {

stroke(0);

fill(0,100);

rect(random(width),random(height),vol*20,vol*20);

}       // Jeśli głośność jest większa niż 0,5, prostokąt jest rysowany w losowej lokalizacji w oknie. Im
        //głośniejsza głośność, tym większy prostokąt

// Wykres ogólnej głośności

// Najpierw narysuj pasek tła

fill(175);

rect(0,0,20,height);

// Następnie narysuj rozmiar prostokąta zgodnie z głośności

fill(0);

rect(0,height-vol*height/2,20,vol*height/2);

}

// Zamknij silnik dźwięk

public void stop() {

Sonia.stop();

super.stop();

}
```

Ta aplikacja działa całkiem dobrze, ale tak naprawdę nie emuluje klapy. Zauważ, że każde klaskanie powoduje narysowanie kilku prostokątów do okna. Dzieje się tak, ponieważ dźwięk, choć wydaje się natychmiastowy dla naszych ludzkich uszu, pojawia się w pewnym okresie czasu. Może to być bardzo krótki okres czasu, ale wystarczy utrzymać poziom głośności powyżej 0,5 przez kilka cykli poprzez draw (). Aby klaska wyzwoliła zdarzenie tylko raz i raz, musimy przemyśleć logikę naszego programu. Mówiąc po polsku, staramy się to osiągnąć:

• Jeśli poziom dźwięku przekracza 0,5, klaszczesz i uruchamiasz zdarzenie. Jednak nie uruchamiaj zdarzenia, jeśli właśnie zrobiłeś to przed chwilą!

Kluczem jest tutaj, jak zdefiniowaliśmy „chwilę temu. „Jednym z rozwiązań byłoby wdrożenie timera, to znaczy, aby wyzwalać zdarzenie tylko raz, a następnie poczekać sekundę, zanim będzie można ponownie uruchomić zdarzenie. To jest całkowicie OK rozwiązanie. Niemniej jednak, z dźwiękiem, zegar jest całkowicie niepotrzebny, ponieważ sam dźwięk powie nam, kiedy skończymy klaskać!

• Jeśli poziom dźwięku jest mniejszy niż 0,25, wtedy jest cicho i zakończyliśmy klaskanie. OK, dzięki tym dwóm elementom logicznym jesteśmy gotowi zaprogramować ten „podwójnie progowany” algorytm.

Istnieją dwa progi, jeden do określenia, czy zaczęliśmy klaskać, a drugi do określenia, czy zakończyliśmy. Będziemy potrzebować zmiennej boolowskiej, aby powiedzieć nam, czy aktualnie klaszczymy, czy nie. Załóżmy, że klaskanie = false.

• Jeśli poziom dźwięku przekracza 0,5 i nie klaskamy, uruchom zdarzenie i ustaw klaskanie = prawda.

• Jeśli klaskamy i poziom dźwięku jest mniejszy niż 0,25, wtedy jest cicho i ustaw klaskanie = fałsz.

W kodzie oznacza to:

```
// Jeśli objętość jest większa niż jeden i nie klaszczymy, narysuj prostokąt
if (vol > 0.5 & & !clapping) {
// Zdarzenie wyzwalające!
clapping = true; // We are now clapping!
} else if (clapping & & vol < 0.25) { // If we are finished clapping
clapping = false;
}
```

Oto pełny przykład, w którym jeden i tylko jeden prostokąt pojawia się na klaśnięcie.

Przykład 20-7: Zdarzenia dźwiękowe (podwójny próg) z Sonia

```
// Importuj bibliotekę Sonia
import pitaru.sonia_v2_9.*;
float clapLevel = 0.5; // How loud is a clap
float threshold = 0.25; // How quiet is silence
boolean clapping = false;
void setup() {
size(200,200);
Sonia.start(this); // Uruchom silnik Sonia.
LiveInput.start(); // rozpocznij słuchanie mikrofonu
smooth();
background(255);
}
void draw() {
// Pobierz ogólną głośność (od 0 do 2.0)
float vol = LiveInput.getLevel();
// Jeśli głośność jest większa niż 0,5 i
// nie klaszczemy, narysuj prostokąt
```

```
if (vol > clapLevel & & !clapping) {          // Jeśli głośność jest większa niż 1.0, a my wcześniej nie
                                              //klaskaliśmy, klaszczemy!
stroke(0);
fill(0,100);
rect(random(width),random(height),vol*20,vol*20);
clapping = true; // Klaszczemy teraz!
// Jeśli zakończyliśmy klaskanie
} else if (clapping & & vol < threshold) { // W przeciwnym razie, gdybyśmy po prostu klaskali, a poziom
                                              //głośności spadł poniżej 0,25, nie będziemy już klaskać!
clapping = false;
}
// Wykres ogólnej głośności
// Najpierw narysuj pasek tła
noStroke();
fill(200);
rect(0,0,20,height);
// Następnie narysuj rozmiar prostokąta zgodnie z głośnością
fill(100);
rect(0,height-vol*height/2,20,vol*height/2);
// Rysuj linie na poziomach progowych
stroke(0);
line(0,height-clapLevel*height/2,19,height-clapLevel*height/2 );
line(0,height-threshold*height/2,19,height-threshold*height/2 );
}
// Zamknij silnik dźwięku
public void stop() {
Sonia.stop();
super.stop();
}
```

Ćwiczenie 20-6: Wyzwól zdarzenie z przykładu 20-7 po sekwencji dwóch klaśnięć. Oto kod do uruchomienia, który zakłada istnienie zmiennej o nazwie „clapCount”.

```
if (vol > c lapLevel & & !clapping) {
```

```
clapCount___;

if (_________________) {

_________________;

_________________;

_________________;

}

_________________;

} else if (clapping & & vol < 0.5) {

clapping = false;

}
```

Ćwiczenie 20-7: Stwórz prostą grę sterowaną głośnością. Sugestia: Najpierw spraw, by gra działała za pomocą myszy, a następnie zastąp mysz myszą na żywo. Niektóre przykłady to Pong, gdzie pozycja wiosła jest związana z objętością, lub Duck Hunt, gdzie kula jest strzelana za każdym razem, gdy użytkownik klaszcze.

Eksportowanie

Skupiliśmy naszą energię na temat uczenia się programowania. Niemniej jednak, w końcu nasze dzieci kodują dorastają i chcą znaleźć drogę na świat. Rozdział ten jest poświęcony omówieniu różnych opcji publikowania dostępnych dla szkiców przetwarzania.

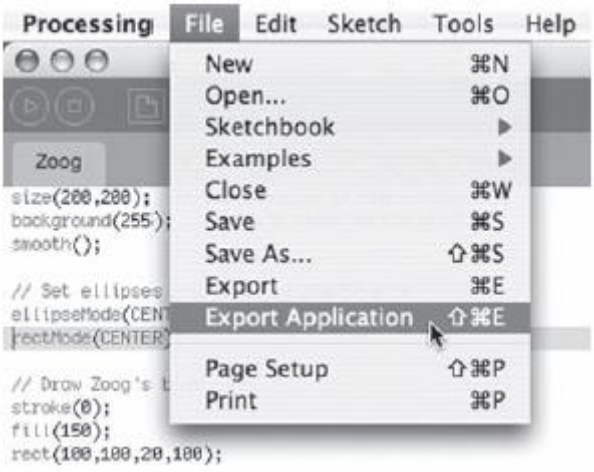
21.1 Aplety internetowe

Szkice przetwarzania można eksportować jako aplety Java, aby uruchomić je w sieci. Jak to działa, omówiono w części 2. Widzieliśmy jednak, że z powodu ograniczeń bezpieczeństwa aplety nie mogą wykonywać pewnych zadań, takich jak połączenie ze zdalnym serwerem lub dostęp do portu szeregowego, kamery internetowej lub innego urządzenia sprzętowego. Jeśli masz szkic, który musisz opublikować online, ale używasz funkcji, która jest ograniczona, istnieją pewne rozwiązania. Jedną z możliwości (jak wspomniano w rozdziale 16) jest podpisanie apletu. Niektóre wskazówki, jak to zrobić, są dostępne na stronie internetowej tej książki (http://www.learningprocessing.com/sandbox).

21.2 Aplikacje samodzielne

Świetną cechą Processing jest to, że szkice mogą być publikowane jako samodzielne aplikacje, co oznacza programy, które można uruchamiać bez uruchamiania środowiska programistycznego. Jeśli tworzysz program dla instalacji lub środowiska, ta funkcja pozwoli Ci tworzyć aplikacje, które mogą być łatwo uruchamiane po uruchomieniu komputera. Aplikacja Eksportuj jest również przydatna, jeśli chcesz uruchomić wiele kopii szkicu w tym samym czasie. Żadne z ograniczeń zabezpieczeń apletu nie ma zastosowania do aplikacji. Aby wyeksportować jako aplikację, przejdź do:

FILE → EXPORT APPLICATION



Zauważysz wtedy, że pojawią się trzy nowe foldery, jak pokazano na rysunku

Przetwarzanie automatycznie tworzy aplikacje dla trzech systemów operacyjnych: Mac OS X, Windows i Linux. Jeśli eksportujesz aplikację z komputera z systemem Windows, aplikacja Mac nie będzie działać poprawnie (instrukcje naprawienia tego problemu znajdują się w pliku readme.txt, który się pojawi). Aby tego uniknąć, wystarczy wyeksportować aplikację na komputer Mac.

Folder będzie zawierał wszystkie potrzebne pliki:

• sketchName.exe (lub nazwany po prostu sketchName na komputerach Mac i Linux). Ten plik jest aplikacją podwójnie klikalną.

• katalog „źródło”. Folder aplikacji będzie zawierał katalog zawierający pliki źródłowe twojego programu. Ten folder nie jest wymagany do uruchomienia aplikacji.

• „lib”. Ten folder pojawia się tylko w systemach Windows i Linux i zawiera wymagane pliki bibliotek. Zawsze będzie zawierał core.jar, rdzeniową bibliotekę przetwarzania, jak również wszelkie inne importowane. W systemie Mac OS X pliki biblioteki są widoczne po kliknięciu i kliknięciu pliku aplikacji i wybraniu opcji „Pokaż zawartość pakietu”. ”

Użytkownik aplikacji będzie potrzebował Javy zainstalowanej na swoim komputerze, aby aplikacja działała poprawnie. Na komputerze Mac Java jest preinstalowana z systemem operacyjnym, więc nie powinieneś mieć wielkich problemów, o ile użytkownik końcowy nie pomylił się przy instalacji Java. W systemie Windows, jeśli jest to problem, możesz dołączyć Javę do aplikacji, kopiując folder „Java” z katalogu Processing do folderu aplikacji (aby to zrobić, musisz być na komputerze). Istnieje kilka sztuczek związanych z funkcją eksportu aplikacji. Na przykład, jeśli chcesz mieć własny tekst na pasku tytułu, możesz dodać następujący kod do setup().

frame.setTitle („Moja super niesamowita aplikacja!”);

Ponadto, chociaż wyeksportowane aplikacje działają w trybie okienkowym (w przeciwieństwie do trybu „obecnego”), można ustawić aplikacje, aby działały na pełnym ekranie, dodając jakiś kod. Zrozumienie kodu wymaga trochę zaawansowanej znajomości języka Java i wewnętrzne działanie szkiców Processing. Niektóre wskazówki na ten temat zostaną ujawnione w części 23, ale na razie kod ten można po prostu skopiować:

```
static public void main (String args []) {

PApplet.main (new String [] {"--present", "SketchName"});

}
```

Ten kod można wstawić w dowolnym miejscu, jako jego własny blok (powyżej setup() jest to dobre miejsce). „Nazwa szkicu” musi być zgodna z nazwą szkicu Processing

Ćwiczenie 21-1: Wyeksportuj samodzielną aplikację dla dowolnego szkicu przetwarzania, który stworzyłeś lub dowolnego przykładu

21.3 Pliki PDF o wysokiej rozdzielczości

Używaliśmy przetwarzania głównie jako środka do tworzenia programów graficznych uruchamianych na ekranie komputera. W rzeczywistości, jeśli przypominasz sobie część 3, spędziliśmy dużo czasu na poznawaniu, jak przepływ programu działa w czasie. Niemniej jednak teraz jest dobry moment, aby powrócić do idei statycznego programu, którego jedynym celem jest stworzenie statycznego obrazu. Biblioteka Przetwarzania PDF pozwala nam wykonywać te statyczne szkice i tworzyć obrazy o wysokiej rozdzielczości do druku. Poniżej przedstawiono wymagane kroki do korzystania z biblioteki PDF.

Krok 1. Importuj bibliotekę.

import processing.pdf. *;

Krok 2. W setup() użyj funkcji size() z trybem „PDF” i argumentem String file.

size(400, 400, PDF, "filename.pdf");

Krok 3. W draw(), zrób swoją magię!

background(255);

fill(175);

stroke(0);

ellipse(width/2,height/2,160,160);

Krok 4. Wywołaj funkcję „exit ()”. To jest bardzo ważne. Wywołanie exit () powoduje zakończenie renderowania pliku PDF. Bez tego plik nie otworzy się poprawnie.

exit(); // Required!

Oto, jak program wygląda razem:

Przykład 21-1: Podstawowy PDF

// mportuj bibliotekę

import processing.pdf.*;

// Używając trybu „PDF”, czwarty argument jest nazwą pliku

size(400, 400, PDF, "filename.pdf");

//Narysuj kilka rzeczy!

background(255);

fill(175);

stroke(0);

ellipse(width/2,height/2,160,160);

```
// Bardzo ważne, wymagane do poprawnego renderowania pliku PDF

exit();
```

Jeśli uruchomisz ten przykład, zauważysz, że nie pojawia się żadne okno. Po ustawieniu trybu renderowania przetwarzania na PDF okno szkicu nie będzie już wyświetlane. Dzieje się tak, ponieważ często używa się trybu PDF do tworzenia złożonych obrazów o bardzo wysokiej rozdzielczości, które nie mogą być łatwo wyświetlane na ekranie. Możliwe jest jednak wyświetlenie okna Szkic przetwarzania podczas renderowania pliku PDF przy użyciu funkcji beginRecord() i endRecord(). Jest uruchamiany wolniej niż pierwszy przykład, ale pozwala zobaczyć, co jest zapisywane.

Przykład 21-2: PDF przy użyciu beginRecord()

```
import processing.pdf.*;

void setup() {

size(400, 400);

beginRecord(PDF, " filename.pdf " );    //beginRecord () uruchamia proces. Pierwszy argument
                                        //powinien odczytywać PDF, a drugi to nazwa pliku.

}

void draw() {

// Narysuj kilka rzeczy!

smooth();

background(100);

fill(0);

stroke(255);

ellipse(width/2,height/2,160,160);

endRecord();            //endRecord() jest wywoływany, aby zakończyć PDF

noLoop();               //Nie ma powodu, aby zapętlać więcej, ponieważ plik PDF jest gotowy

}
```

endRecord () nie musi być wywoływany w pierwszej renderowanej ramce, dlatego ten tryb może być użyty do wygenerowania pliku PDF skompilowanego z wielu cykli poprzez draw (). Poniższy przykład wykonuje program „Scribbler" z części 16 i wyświetla wynik w formacie PDF. Tutaj kolory nie są pobierane ze strumienia wideo, ale są wybierane na podstawie zmiennej licznika.

Przykład 21-3: Wiele ramek w jednym pliku PDF

```
import processing.pdf.*;

float x = 0;

float y = 0;

void setup() {
```

```
size(400, 400);

beginRecord(PDF, "scribbler.pdf");

background(255);        //background() powinno znajdować się w setup (). Jeśli tło () zostanie
                        //umieszczone w draw(), plik PDF zgromadzi wiele elementów graficznych,
                        //aby je wymazywać.

}

void draw() {

// Wybierz nowy x i y

float newx = constrain(x + random( -20,20),0,width);

float newy = constrain(y + random( -20,20),0,height);

// Narysuj linię od x, y do newx, newy

stroke(frameCount%255,frameCount*3%255,

frameCount*11%255,100);

strokeWeight(4);

line(x,y,newx,newy);

// Zapisz newx, newy w x, y

x = newx;

y = newy;

}

// Po naciśnięciu myszy kończymy PDF

void mousePressed() {

endRecord();            //W tym przykładzie użytkownik wybiera, kiedy zakończyć renderowanie pliku
                        //PDF, klikając myszą.

// Możemy powiedzieć, że Processing otworzy plik PDF

open(sketchPath( "scribbler.pdf"));

noLoop();

}
```

Jeśli renderujesz kształty 3D (w trybie P3D lub OPENGL), będziesz chciał użyć beginRaw () i endRaw () zamiast beginRecord () i endRecord (). Ten przykład używa również zmiennej logicznej („RecordPDF”).

Przykład 21-4: PDF i openGL

```
// Używamy OPENGL

import processing.opengl.*;
```

```
import processing.pdf.*;

// Obrót kostki

float yTheta = 0.0;

float xTheta = 0.0;

// Aby uruchomić nagrywanie pliku PDF

boolean recordPDF = false;      //Zmienna logiczna, która po ustawieniu na wartość true powoduje
                                //utworzenie pliku PDF.

void setup() {

size(400, 400, OPENGL);         //Tryb OPENGL lub P3D wymaga użycia beginRaw () i endRaw ()
                                //zamiast beginRecord () i endRecord ()

smooth();

}

void draw() {

// Rozpocznij tworzenie pliku PDF

if (recordPDF) {

beginRaw(PDF, " 3D.pdf " );     //Jeśli umieścisz „####” w nazwie pliku - „3D - ####. Pdf” - oddzielne,
                                //ponumerowane pliki PDF zostaną utworzone dla każdej
                                //renderowanej ramki.

}

background(255);

stroke(0);

noFill();

translate(width/2,height/2);

rotateX(xTheta);

rotateY(yTheta);

box(100);

xTheta + = 0.02;

yTheta + = 0.03;

// Zakończ tworzenie pliku PDF

if (recordPDF) {

endRaw();

recordPDF = false;
```

```
}

}

// Zrób plik PDF po naciśnięciu myszy

void mousePressed() {

recordPDF = true;

}
```

Dwie ważne uwagi dotyczące biblioteki PDF.

• Images - jeśli wyświetlasz obrazy w pliku PDF, niekoniecznie będą one wyglądać dobrze po wyeksportowaniu. Obraz o wymiarach 320 x 240 pikseli jest nadal obrazem o rozdzielczości 320 x 240 pikseli, niezależnie od tego, czy jest renderowany w wysokiej rozdzielczości PDF lub nie.

• Text - jeśli wyświetlasz tekst w pliku PDF, musisz mieć zainstalowaną czcionkę, aby poprawnie wyświetlić plik PDF. Jednym ze sposobów na to jest włączenie „textMode (SHAPE); ”Po rozmiarze (). Spowoduje to wyświetlenie tekstu jako kształtu pliku PDF i nie wymaga zainstalowanej czcionki.

Pełna dokumentacja biblioteki PDF znajduje się na stronie referencyjnej Przetwarzanie pod adresem http://processing.org/reference/libraries/pdf/index.html. Chociaż biblioteka PDF zajmie się wieloma potrzebami związanymi z generacją w wysokiej rozdzielczości, istnieją dwie inne biblioteki, które mogą być interesujące. Jednym z nich jest proSVG Christiana Riekoffa za eksport plików w formacie SVG („Scalable Vector Graphics”):

http://www.texone.org/prosvg/. Innym jest SimplePostScript autorstwa Mariusa Watza do pisania plików wektorowych w formacie PostScript: http://processing.unlekker.net/SimplePostscript/.

Ćwiczenie 21-2: Utwórz plik PDF z dowolnego utworzonego szkicu przetwarzania lub dowolnego przykładu

21.4 Obrazy / saveFrame ()

Pliki PDF o wysokiej rozdzielczości są przydatne do drukowania; jednakże możesz także zapisać zawartość okna Przetwarzania jako plik obrazu (z taką samą rozdzielczością, jak rozmiar samego okna). Jest to realizowane za pomocą save() lub saveFrame().

save() pobiera jeden argument, nazwę pliku obrazu, który chcesz zapisać. save() wygeneruje pliki obrazów w następujących formatach: JPG, TIF, TGA lub PNG, wskazane przez rozszerzenie pliku, które zawierasz w nazwie pliku. Jeśli żadne rozszerzenie nie jest włączone, przetwarzanie będzie domyślnie w formacie TIF.

```
background(255,0,0);

save ("file.jpg");
```

Jeśli wywołasz save () wiele razy z tą samą nazwą pliku, zastąpi on poprzedni obraz. Jeśli jednak chcesz zapisać sekwencję obrazów, funkcja saveFrame () automatycznie numeruje pliki. Przetwarzanie spowoduje wyszukanie ciągu „####” w nazwie pliku i zastąpienie go ponumerowaną sekwencją obrazów.

```
void draw() {
```

```
background(random(255));

saveFrame( "file####.jpg");

}
```

Ta technika jest powszechnie używana do importowania ponumerowanej sekwencji obrazów do oprogramowania wideo lub animacji.

21.5 MovieMaker

Sekwencja obrazów, która powstała w wyniku operacji saveFrame (), może zostać przekształcona w plik filmowy za pomocą QuickTime Pro, iMovie lub dowolnej liczby pakietów oprogramowania wideo. Jednak w przypadku dłuższych filmów posiadanie katalogu pełnego tysięcy zdjęć może okazać się nieco nieporęczne. Użycie klasy MovieMaker ułatwia ten proces, zapisując ramki bezpośrednio do pliku filmowego. Pierwotnie stworzyłem MovieMaker jako bibliotekę, do której się przyczyniłem, ale jest ona teraz dostępna w samej bibliotece wideo Processing.

```
import processing.video. *;
```

Obiekt MovieMaker wymaga następujących argumentów (w kolejności):

• this —Odnosi się do szkicu, z którym będzie skojarzony film.

• width - liczba całkowita, zazwyczaj taka sama szerokość jak szkic.

• height - liczba całkowita, zazwyczaj taka sama wysokość jak szkic.

• filename - ciąg znaków dla nazwy pliku filmu.

• framerate - szybkość klatek dla filmu, zwykle 30.

• type - jaki format kompresji powinien być używany w filmie.

• quality - jakość kompresji dla filmu.

Przykład konstruktora jest następujący:

```
mm = new MovieMaker (this, width, height, "test.mov", 30, MovieMaker.H263, MovieMaker.HIGH);
```

Biblioteka udostępnia wiele opcji dla kodeków kompresji. Termin kodek pochodzi pierwotnie od terminu „dekoder kodera”, odnosząc się do urządzenia sprzętowego, które konwertuje analogowy sygnał wideo na cyfrowy. Tutaj jednak kodek odnosi się do „kompresora-dekompresora”, oprogramowania, które ukrywa wideo między jego surową, nieskompresowaną formą (do oglądania) i jego skompresowaną formą (do przechowywania). Poniżej znajdują się kodery-dekodery dostępne w bibliotece MovieMaker (sprawdź stronę referencyjną w poszukiwaniu aktualizacji tej listy: http://processing.org/reference/libraries/video/MovieMaker.html).

ANIMATION, BASE, BMP, CINEPAK, COMPONENT, CMYK, GIF, GRAPHICS, JPEG, MS_VIDEO, MOTION_JPEG_A, MOTION_JPEG_B, RAW, SORENSON, VIDEO, H261, H263, H264

Wybór kodeka wpłynie zarówno na rozmiar pliku filmowego, jak i jakość. RAW, na przykład, przechowa wideo w jego surowej nieskompresowanej formie (a zatem najwyższej jakości), ale spowoduje powstanie ogromnego pliku wideo. Więcej informacji na temat różnych kodeków można znaleźć w witrynie Apple QuickTime.

• QuickTime: http://www.apple.com/quicktime/.

• QuickTime Developer: http://developer.apple.com/quicktime/.

Po wybraniu kodeka należy określić jakość kodeka, od najgorszego do bezstratnego.

WORST, LOW, MEDIUM, HIGH, BEST, LOSSLESS

Po skonstruowaniu obiektu MovieMaker ramki można dodawać pojedynczo do filmu, wywołując funkcję addFrame ().

```
void draw () {

// Kilka fajnych rzeczy, które rysujesz!

mm.addFrame ();

}
```

Wreszcie, aby plik filmowy mógł być poprawnie odtwarzany, musi zostać „zakończony" funkcją fi nishMovie (). Jeśli zamkniesz szkic Przetwarzania przed wywołaniem tej funkcji, plik filmu nie zostanie rozpoznany przez QuickTime! Jednym z rozwiązań jest zakończenie filmu kliknięciem myszy. Jeśli jednak używasz myszy już w swoim aplecie, możesz rozważyć inne opcje, takie jak kończenie naciskania klawisza lub po wyczerpaniu się zegara.

```
public void mousePressed () {

mm.finishMovie ();

}
```

Przykład 21-5 rejestruje film z rysunku myszy na ekranie. Film zostaje zakończony po naciśnięciu spacji.

Przykład 21-5: Tworzenie filmu QuickTime

```
import processing.video.*;        //Klasa Movie Maker jest częścią biblioteki wideo Processing

MovieMaker mm; // Zadeklaruj obiekt MovieMaker

void setup() {

size(320, 240);

// Utwórz obiekt MovieMaker o rozmiarze, nazwie pliku

// szybkość odtwarzania, kodek kompresji i jakość

mm = new MovieMaker(this, width, height, "drawing.mov", 30, MovieMaker.H263,

MovieMaker.HIGH);

background(255);

}

void draw() {

stroke(0);

strokeWeight(4);

if (mousePressed) {
```

```
line(pmouseX, pmouseY, mouseX, mouseY);

}

mm.addFrame(); // Dodaj piksele okna do film

}        //Nowa klatka jest dodawana do filmu w każdym cyklu poprzez draw()

void keyPressed() {

if (key = = ' ') {

println( "finishing movie ");

mm.finish(); // Zakończ film, jeśli wciśnięty jest spacja!

}        //Nie zapomnij ukończyć filmu! W przeciwnym razie nie będzie prawidłowo odtwarzany.

}
```

Ćwiczenie 21-3: Stwórz film z dowolnego szkicu przetwarzania, który stworzyłeś lub dowolnego przykładu

Projekt Lekcji Dziewięć

Wybierz jedną lub obie!

1. Włącz dźwięk do szkicu przetwarzania, dodając efekty dźwiękowe lub sterując szkicem na żywo.

2. Użyj przetwarzania, aby wygenerować wynik inny niż grafika w czasie rzeczywistym. Zrób wydruk (używając PDF). Utwórz film (używając MovieMaker) i tak dalej.

Użyj miejsca podanego poniżej, aby naszkicować projekty, notatki i pseudokod dla twojego projektu.

Zaawansowane programowanie obiektowe

W części 8 wprowadziliśmy programowanie obiektowe („OOP"). Główną zasadą rozdziału było połączenie danych i funkcjonalności w jeden pomysł, klasę. Klasa jest szablonem iz tego szablonu tworzyliśmy instancje obiektów i zapisywaliśmy je w zmiennych i tablicach. Chociaż nauczyliśmy się pisać klasy i tworzyć obiekty, nie zagłębiliśmy się w podstawowe zasady OOP i zbadaliśmy jego zaawansowane funkcje. Teraz, gdy zbliżamy się do końca (i wkrótce wkroczymy w świat Javy w następnej części), jest to dobry moment, aby przemyśleć przeszłość i podjąć kroki w kierunku przyszłości.

Programowanie obiektowe w przetwarzaniu i Javie jest definiowane przez trzy podstawowe pojęcia: enkapsulacja, dziedziczenie i polimorfizm. Znamy już enkapsulację; po prostu nie sformalizowaliśmy naszego rozumienia pojęcia i użyliśmy terminologii. Dziedziczenie i polimorfizm to zupełnie nowe koncepcje, które omówimy w tym rozdziale. (Na końcu tej części przyjrzymy się również szybko przeładowaniu metod, które pozwala obiektom na więcej niż jeden sposób wywoływania konstruktora).

22.1 Hermetyzacja

Aby zrozumieć enkapsulację, wróćmy do przykładu klasy samochodów. I weźmy ten przykład na świat i pomyślmy o prawdziwym obiekcie samochodowym, obsługiwanym przez prawdziwego kierowcę: ty. To miły letni dzień i jesteś zmęczony programowaniem i decydujesz się na plażę. weekend. Zezwolenie na ruch, masz nadzieję na przyjemną jazdę, gdzie obrócisz kierownicę kilka razy, wciśniesz gaz i hamulce, i zatrzasniesz pokrętło w radiu. Ten samochód, który prowadzisz, jest zamknięty. Jedyne, co musisz zrobić, to sterować funkcjami: steruj (), gaz (), hamulec () i radio (). Czy wiesz, co jest pod maską? Jak konwerter katalityczny łączy się z silnikiem lub jak silnik łączy się z chłodnicą międzystopniową? Co robią zawory i przewody oraz przekładnie i pasy? Oczywiście, jeśli jesteś doświadczonym mechanikiem samochodowym, możesz być w stanie odpowiedzieć na te pytania, ale chodzi o to, że nie musisz jeździć samochodem. To jest hermetyzacja. Enkapsulacja jest definiowana jako ukrywanie wewnętrznych działań obiektu przed użytkownikiem tego obiektu. Jeśli chodzi o programowanie obiektowe, „wewnętrzne działanie" obiektu to dane (zmienne tego obiektu) i funkcje. „Użytkownikiem" obiektu jest ty, programista, który tworzy instancje obiektów i używa ich w całym kodzie. Dlaczego to dobry pomysł? W części 8 (i wszystkich przykładach OOP) podkreślono zasady modułowości i możliwości ponownego wykorzystania. Oznacza to, że jeśli już wiesz, jak zaprogramować samochód, po co robić to ciągle i za każdym razem, gdy musisz zrobić samochód? Po prostu zorganizuj wszystko w klasie samochodów, a zaoszczędzisz sobie wielu bólów głowy. Enkapsulacja idzie o krok dalej. OOP nie tylko pomaga uporządkować kod, ale także chroni przed popełnianiem błędów. Jeśli nie będziesz bał się okablowania samochodu podczas prowadzenia pojazdu, jest mniej prawdopodobne, że złamiesz samochód. Oczywiście czasami samochód psuje się i musisz go naprawić, ale wymaga to otwarcia kaptura i obejrzenia kodu wewnątrz samej klasy. Weź następujący przykład. Powiedzmy, że piszesz klasę konta bankowego, która ma punkt flotacyjny

```
BankAccount account = new BankAccount(1000);      //Obiekt konta bankowego z saldem
                                                  //początkowym w wysokości 1000 USD.
```

Będziesz chciał zamknąć tę równowagę i ukryć ją. Czemu? Powiedzmy, że musisz wypłacić pieniądze z tego konta. Więc odejmujesz 100 $ od tego konta.

```
account.balance = account.balance – 100      //Wypłata 100 USD poprzez bezpośredni dostęp do
                                             //zmiennej salda
```

Ale co, jeśli istnieje opłata za wypłatę pieniędzy? Powiedz, 1,25 $? Szybko przestaniemy korzystać z naszej oferty programowania bankowego, pozostawiając te szczegóły. Dzięki enkapsulacji zachowalibyśmy nasze zadanie, zamiast tego napisali poniższy kod.

```
account.withdraw(100);          //Wypłata 100 $ przez wywołanie metody!
```

Jeśli funkcja withdraw() zostanie napisana poprawnie, nigdy nie zapomnimy opłaty, ponieważ będzie ona występować za każdym razem, gdy metoda zostanie wywołana.

```
void withdraw(float amount) {

float fee = 1.25;          //Funkcja ta zapewnia, że opłata jest również potrącana, gdy pieniądze są
                           //wypłacane

account – = (amount + fee);

}
```

Kolejną zaletą tej strategii jest to, że jeśli bank zdecyduje się podnieść opłatę do 1,50 USD, może po prostu dostosować zmienną opłaty w klasie Konto bankowe i wszystko będzie działało! Z technicznego punktu widzenia, aby postępować zgodnie z zasadami enkapsulacji, nigdy nie można uzyskać bezpośredniego dostępu do zmiennych wewnątrz klasy i można je pobrać tylko za pomocą metody. Dlatego często widzicie programistów używających wielu funkcji znanych jako gettery i settery, funkcji, które pobierają lub zmieniają wartości zmiennych. Oto przykład klasy Point z dwiema zmiennymi (xiy), do których dostęp uzyskuje się za pomocą getterów i setterów.

```
class Point {

float x,y;

Point(float tempX, float tempY) {

x = tempX;

y = tempY;

}

// Getters

float getX() {     //Zmienne x i y są dostępne za pomocą funkcji getter i setter.

return x;

}

float getY() {

return y;

}

// Setters

float setX(float val) {

x = val;

}
```

```
float setY(float val) {

if (val > height) val = height;     //Gettery i settery pozwalają nam chronić wartość zmiennych. Na
                                    //przykład tutaj wartość y nigdy nie może być ustawiona na większą
                                    //niż wysokość szkicu.

y = val;

}

}
```

Gdybyśmy chcieli zrobić punkt i zwiększyć wartość y, musielibyśmy to zrobić w ten sposób:

```
p.setY (p.getY () + 1);     //Zamiast: p.y = p.y + 1;
```

Java faktycznie pozwala nam oznaczyć zmienną jako „prywatną", aby uniemożliwić bezpośredni dostęp do niej (innymi słowy, jeśli spróbujesz, program nawet nie uruchomi się).

```
class Point {

private float x;  //Chociaż rzadko są one widoczne w przykładach przetwarzania, zmienne można
                  //ustawić jako prywatne, co oznacza, że są dostępne tylko wewnątrz samej klasy.
                  //Domyślnie wszystkie zmienne i funkcje w przetwarzaniu są „publiczne".

private float y;

}
```

Podczas gdy enkapsulacja jest podstawową zasadą programowania obiektowego i jest bardzo ważna przy projektowaniu aplikacji na dużą skalę programowanych przez zespół programistów, trzymanie się litery prawa (jak przy zwiększaniu wartości y obiektu Point powyżej) jest często raczej niewygodne i prawie głupie dla prostego szkicu przetwarzania. Więc nie jest koniec świata, jeśli utworzysz klasę Point i uzyskasz dostęp do zmiennych x i y bezpośrednio. Zrobiliśmy to kilka razy w przykładach. Jednak zrozumienie zasady enkapsulacji powinno być siłą napędową w sposobie projektowania obiektów i zarządzania kodem. Za każdym razem, gdy zaczynasz ujawniać wewnętrzne działanie obiektu poza samą klasą, powinieneś zadać sobie pytanie: czy to jest konieczne? Czy ten kod może wchodzić do funkcji wewnątrz klasy? W końcu będziesz szczęśliwszym programistą i utrzymasz tę pracę w banku.

22.2 Dziedziczenie

Dziedziczenie, drugie z naszej listy trzech podstawowych pojęć programowania obiektowego, pozwala nam tworzyć nowe klasy oparte na istniejących klasach. Spójrzmy na świat zwierząt: psów, kotów, małp, pand, wombatów i pokrzyw morskich. Zasadniczo zacznijmy od programowania klasy Dog. Obiekt Dog będzie miał zmienną wieku (liczbę całkowitą), a także funkcje eat(), sleep() i bark().

```
class Dog {

int age;

Dog() {

age = 0;

}

void eat() {
```

```
// eating code goes here

}

void sleep() {

// sleeping code goes here

}

void bark() {

println( " WOOF! " );

}

}
```

Skończone z psami, możemy teraz przejść do kotów.

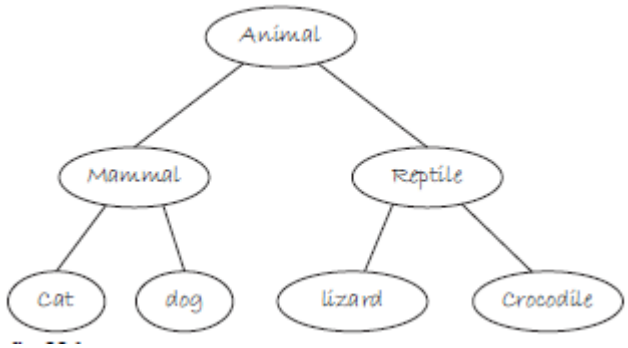
```
class Cat {       //Zauważ, że psy i koty mają te same zmienne (wiek) i funkcje (jedzenie, sen). Mają
                  //jednak także wyjątkową funkcję na szczekanie lub miauczenie.

int age;

Cat() {

age = 0;

}

void eat() {

// eating code goes here

}

void sleep() {

// sleeping code goes here

}

void meow() {

println( " MEOW! " );

}

}
```

Niestety, gdy przechodzimy na ryby, konie, niedźwiedzie koala i lemury, proces ten stanie się dość nudny, ponieważ wciąż przepisujemy ten sam kod. Co by było, gdybyśmy mogli stworzyć ogólną klasę ANiestety, gdy przechodzimy na ryby, konie, niedźwiedzie koala i lemury, proces ten stanie się dość nudny, ponieważ wciąż przepisujemy ten sam kod. Co by było, gdybyśmy mogli stworzyć ogólną klasę zwierząt, aby opisać każdy rodzaj zwierzęcia? Przecież wszystkie zwierzęta jedzą i śpią. Moglibyśmy wtedy powiedzieć co następuje:

• Pies jest zwierzęciem i posiada wszystkie właściwości zwierząt i potrafi robić wszystko, co potrafią zwierzęta. Ponadto pies może szczekać.

• Kot jest zwierzęciem i posiada wszystkie właściwości zwierząt i potrafi robić wszystko, co potrafią zwierzęta. Ponadto kot może miauczeć.

Dziedziczenie pozwala nam zaprogramować właśnie to. W przypadku dziedziczenia klasy mogą dziedziczyć właściwości (zmienne) i funkcjonalność (metody) z innych klas. Klasa Dog to dziecko (AKA podklasa) klasy Animal. Dzieci dziedziczą wszystkie zmienne i funkcjonują w sposób nieistotny względem rodzica (superklasa AKA). Dzieci mogą również zawierać dodatkowe zmienne i funkcje, których nie ma w rodzicach. Dziedzictwo następuje po strukturze (podobnie jak filogenetyczne „drzewo życia"). Psy mogą dziedziczyć po psach, które dziedziczą po ssakach, które dziedziczą po zwierzętach i tak dalej.



Oto jak działa składnia z dziedziczeniem.

```
class Animal {
int age;          //Klasa Animal jest klasą nadrzędną (lub super).
Animal() {
age = 0;
}                 //Zmienny wiek i funkcje eat () i sleep () są dziedziczone przez psa i kota.
void eat() {
// kod jedzenie jest tu
}
void sleep() {
// kod spania jest tu
}
}
class Dog extends Animal {    //Klasa Dog jest klasą podrzędną (lub podrzędną). Jest to oznaczone
                              //kodem „extends Animal"
Dog() {
super();
}
void bark() {
```

```
println( "WOOF!");

}

}

class Cat extends Animal {

Cat() {

super();          //super() oznacza wykonanie kodu znalezionego w klasie nadrzędnej.

}

void bark() {     //Ponieważ bark () nie jest częścią klasy nadrzędnej, musimy zdefiniować ją w klasach
                  //potomnych

println( " MEOW! " );

}

}
```

Wprowadzono następujące nowe warunki:

• extends - to słowo kluczowe służy do wskazania klasy nadrzędnej dla definiowanej klasy. Zauważ, że klasy mogą rozszerzać tylko jedną klasę. Jednak klasy mogą rozszerzać klasy, które rozszerzają inne klasy, to znaczy, Dog rozszerza Animal, Terrier rozszerza Dog. Wszystko jest dziedziczone aż do końca.

• super()-super wywołuje konstruktor w klasie nadrzędnej. Innymi słowy, cokolwiek robisz w konstruktorze nadrzędnym, zrób to również w konstruktorze potomnym. Nie jest to wymagane, ale jest dość powszechne (zakładając, że obiekty potomne mają być tworzone w taki sam sposób, jak ich rodzice). Inny kod można zapisać w konstruktorze oprócz super ().

Podklasę można rozszerzyć tak, aby zawierała dodatkowe funkcje i właściwości poza tym, co jest zawarte w nadklasie. Załóżmy na przykład, że obiekt Dog ma zmienną koloru włosów oprócz wieku, która jest ustawiana losowo w konstruktorze. Klasa wyglądałaby teraz tak:

```
class Dog extends Animal {

color haircolor; //Klasa potomna może wprowadzać nowe zmienne, które nie są uwzględniane w
                 //elemencie macierzystym.

Dog() {

super();

haircolor = color(random(255));

}

void bark() {

println( " WOOF! " );

}

}
```

Zauważ, jak konstruktor nadrzędny jest wywoływany przez super(), ustawiając wiek na 0, ale kolor włosów jest ustawiany wewnątrz samego konstruktora Dog. Załóżmy, że obiekt Dog zjada ę inaczej niż ogólne zwierzę. Funkcje nadrzędne mogą zostać zastąpione przez przepisanie funkcji wewnątrz podklasy.

```
class Dog extends Animal {

color haircolor;

Dog() {

super();

haircolor = color(random(255));

}

void eat() {

// Kod określający, w jaki sposób pies szczególnie je

                    //Dziecko może w razie potrzeby zastąpić funkcję nadrzędną

}

void bark() {

println( " WOOF! " );

}

}
```

Ale co, jeśli pies powinien jeść w taki sam sposób, w jaki robi to zwierzę, ale z pewną dodatkową funkcjonalnością? Podklasa może zarówno uruchomić kod z klasy nadrzędnej, jak i wprowadzić kod niestandardowy.

```
class Dog extends Animal {

color haircolor;

Dog() {

super();

haircolor = color(random(255));

}

void eat() {

// Wywołaniel eat() z Animal    //Dziecko może wykonać funkcję od rodzica podczas dodawania
                                //własnego kodu.

super.eat();

// Dodaj dodatkowy kod

// za to, jak pies szczególnie je
```

```
println( "Yum!!!");

}

void bark() {

println( "WOOF!");

}

}
```

Ćwiczenie 22-1: Kontynuując przykład samochodu z naszej dyskusji o hermetyzacji, w jaki sposób zaprojektowałbyś system klas pojazdów (tj. Samochodów, ciężarówek, autobusów i motocykli)? Jakie zmienne i funkcje zostałyby uwzględnione w klasie nadrzędnej? A co zostanie dodane lub nadpisane w klasach podrzędnych? Co jeśli chciałbyś włączyć samoloty, pociągi i łodzie w tym przykładzie?

22.3 Przykład dziedziczenia: KSZTAŁTY

Teraz, gdy mamy wprowadzenie do teorii dziedziczenia i jej składni, możemy opracować działający przykład w przetwarzaniu. Typowy przykład dziedziczenia obejmuje kształty. Chociaż jest to trochę banalne, jest użyteczne ze względu na swoją prostotę. Stworzymy ogólną klasę „Shape", w której wszystkie obiekty Shape mają położenie x, y, a także rozmiar i funkcję wyświetlania. Kształty poruszają się po ekranie przez „brzęczące" losowe

```
class Shape {

float x;

float y;

float r;

Shape(float x_, float y_, float r_) {

x = x_;

y = y_;

r = r_;

}

void jiggle() {

x + = random(–1,1);

y + = random(–1,1);

}

void display() {  //Ogólny kształt tak naprawdę nie wie, jak się wyświetlać. To będzie być nadpisane w
                  //klasach potomnych.

point(x,y);

}

}
```

Następnie tworzymy podklasę z Shape (nazwijmy ją „Square”). Odziedziczy wszystkie zmienne instancji i metody z Shape. Piszemy nowy konstruktor o nazwie „Square” i wykonujemy kod z klasy nadrzędnej, wywołując super ().

```
class Square extends Shape {

// moglibyśmy dodać zmienne tylko dla Kwadratów, jeśli tego pragniemy

//Zmienne są dziedziczone od rodzica

Square(float x_, float y_, float r_) {

super(x_,y_,r_);        //Jeśli konstruktor nadrzędny przyjmuje argumenty, super() musi przekazać te
                        //argumenty.

}

// Dziedziczy jiggle () od rodzica

// Dodaj metodę wyświetlania

void display() {        //Aha, kwadrat zastępuje jego rodzica do wyświetlenia

rectMode(CENTER);

fill(175);

stroke(0);

rect(x,y,r,r);

}

}
```

Zauważ, że jeśli wywołamy konstruktor nadrzędny za pomocą super (), musimy dołączyć wymagane argumenty. Ponadto, ponieważ chcemy wyświetlić kwadrat na ekranie, zastępujemy wyświetlanie (). Mimo że chcemy, aby kwadrat działał, nie musimy pisać funkcji jiggle(), ponieważ jest ona dziedziczona. Co jeśli chcemy, aby podklasa Shape zawierała dodatkowe funkcje? Poniżej znajduje się przykład klasy Circle, która oprócz rozszerzania Shape zawiera zmienną instancji do śledzenia koloru. (Zauważ, że jest to czysta demonstracja tej cechy dziedziczenia, bardziej logiczne byłoby umieszczenie zmiennej koloru w nadrzędnej klasie Shape.) Rozszerza również funkcję jiggle(), aby dostosować rozmiar i zawiera nową funkcję zmiany koloru.

```
class Circle extends Shape {

//dziedziczy wszystkie zmienne instancji z rodzica + dodając jeden

color c;

Circle(float x_, float y_, float r_, color c_) {

super(x_,y_,r_); // wywołanie konstruktora nadrzędnego

c = c_; // zajmij się także tą nową zmienną instancji

}

//  wywołaj rodzicielską jiggle, ale rób też więcej rzeczy
```

```
void jiggle() {
super.jiggle();
r + = random(-1,1);        //Okrąg porusza się zarówno pod względem rozmiaru, jak i położenia x, y.
r = constrain(r,0,100);
}
void changeColor() {      //Funkcja changeColor () jest unikalna dla okręgów
c = color(random(255));
}
void display() {
ellipseMode(CENTER);
fill(c);
stroke(0);
ellipse(x,y,r,r);
}
}
```

Aby wykazać, że dziedziczenie działa, oto program, który tworzy jeden obiekt Square i jeden obiekt Circle. Klasy Shape, Square i Circle nie są ponownie uwzględniane, ale są identyczne z powyższymi.

Przykład 22-1: Dziedziczenie

```
// Programowanie zorientowane obiektowo pozwala nam definiować klasy w kategoriach innych klas.
// Klasa może być podklasą (aka „child") super klasy (aka „parent".
//  To prosty przykład demonstrujący tę koncepcję, znany jako „dziedziczenie". "
Square s;        //Ten szkic zawiera jeden obiekt Circle i jeden obiekt Square. Nie utworzono obiektu
                 //Shape. Klasa Shape po prostu funkcjonuje jako część naszego drzewa dziedziczenia!
Circle c;
void setup() {
size(200,200);
smooth();
//Kwadrat i okrąg
s = new Square(75,75,10);
c = new Circle(125,125,20,color(175));
}
void draw() {
```

```
background(255);

c.jiggle();

s.jiggle();

c.display();

s.display();

}
```

Ćwiczenie 22-2: Napisz klasę linii, która rozszerza Shape i ma zmienne dla dwóch punktów linii. Gdy linia będzie się poruszać, przesuń oba punkty. Nie będziesz potrzebował „r” do niczego w klasie linii (ale do czego mógłbyś go użyć?).

```
class Line _______ {

float x2,y2;

Line(_______,_______,_______,_______) {

super(____________________);

x2 = _______;

y2 = _______;

}

void jiggle() {

_________________________

_________________________

_________________________

}

void display() {

stroke(255);

line(_____________________________);

}

}
```

Ćwiczenie 22-3: Czy któryś ze stworzonych przez ciebie szkiców zasługuje na wykorzystanie dziedziczenia? Spróbuj znaleźć i zrewidować.

22.4 Polimorfizm

Uzbrojeni w koncepcje dziedziczenia, możemy zaprogramować różnorodne królestwo zwierząt, w którym krążyły szyki psów, kotów, żółwi i kiwi.

```
Dog[] dogs = new Dog[100];
```

```
Cat[] cats = new Cat[101];          //100 psów. 101 kotów. 23 żółwie. 6 kiwi

Turtle[] turtles = new Turtle[23];

Kiwi[] kiwis = new Kiwi[6];

for (int i = 0; i < dogs.length; i + + ) {

dogs[i] = new Dog();

}

for (int i = 0; i < cats.length; i + + ) {

cats[i] = new Cat();        //Ponieważ tablice mają różne rozmiary, potrzebujemy osobnej pętli dla każdej
                            //tablicy

}

for (int i = 0; i < turtles.length; i + + ) {

turtle[i] = new Turtle();

}

for (int i = 0; i < kiwis.length; i + + ) {

kiwis[i] = new Kiwi();

}
```

Wraz z początkiem dnia zwierzęta są bardzo głodne i chcą jeść. Czas na zapętlenie.

```
for (int i = 0; i <dogs.length; i + +) {

psy [i] .eat ();

}

for (int i = 0; i <cats.length; i + +) {

koty [i] .eat ();

}

for (int i = 0; i <turtles.length; i + +) {

żółwie [i] .eat ();

}

for (int i = 0; i <kiwis.length; i + +) {

kiwi [i] .eat ();

}
```

To działa świetnie, ale ponieważ nasz świat rozszerza się o wiele innych gatunków zwierząt, utkniemy w pisaniu wielu pojedynczych pętli. Czy to nie jest konieczne? W końcu wszystkie stworzenia są

zwierzętami i wszystkie lubią jeść. Dlaczego nie mieć tylko jednej tablicy obiektów Animal i wypełnić ją wszystkimi różnymi rodzajami zwierząt?

```
Animal[] kingdom = new Animal[1000]; //Tablica jest typu Animal, ale elementy, które umieściliśmy w
                                     //tablicy, to Psy, Koty, Żółwie i Kiwi.

for (int i = 0; i < kingdom.length; i + + ) {

if (i < 100) kingdom[i] = new Dog();

else if (i < 400) kingdom[i] = new Cat();

else if (i < 900) kingdom[i] = new Turtle();

else kingdom[i] = new Kiwi();

}

for (int i = 0; i < kingdom.length; i + + ) {     //Kiedy nadejdzie czas, w którym wszystkie zwierzęta
                                                  //będą jeść, możemy po prostu zapętlić tę jedną dużą
                                                  //tablicę.

kingdom[i].eat();

}
```

Możliwość traktowania obiektu Dog jako członka klasy Dog lub klasy Animal (jego rodzica) jest znana jako polimorfizm, trzecia zasada programowania obiektowego. Polimorfizm (z greckiego, polymorphos, oznaczający wiele form) odnosi się do traktowania pojedynczej instancji obiektu w wielu formach. Pies jest z pewnością psem, ale ponieważ pies rozszerza zwierzę, można go również uznać za zwierzę. W kodzie możemy odnieść się do niego w obie strony

```
Dog rover = new Dog ();

Animal spot = new Dog ();     //Zazwyczaj typ po lewej stronie musi odpowiadać typowi po prawej
                              //stronie. Z polimorfizmem jest OK, dopóki typ po prawej stronie jest
                              //dzieckiem typu po lewej stronie.
```

Chociaż drugi wiersz kodu może początkowo naruszać reguły składniowe, oba sposoby deklarowania obiektu Dog są legalne. Mimo że deklarujemy spot jako zwierzę, naprawdę tworzymy obiekt Dog i przechowujemy go w zmiennej spot. Możemy bezpiecznie wywołać wszystkie metody na miejscu, ponieważ zasady dziedziczenia wymagają, aby pies mógł zrobić wszystko, co może zwierzę. Co jednak zrobić, jeśli klasa Dog zastąpi funkcję eat () w klasie Animal? Nawet jeśli spot zostanie zadeklarowany jako Animal, Java określi, że jego prawdziwą tożsamością jest Dog i uruchom odpowiednią wersję funkcji eat (). Jest to szczególnie przydatne, gdy mamy tablicę. Przepiszmy przykład Shape z poprzedniej sekcji, aby uwzględnić wiele obiektów Circle i wiele obiektów Square.

```
// Wiele kwadratów i wiele okręgów     //Stary sposób „bez polimorfizmu” z wieloma tablicami.

Square[] s = new Square[10];

Circle[] c = new Circle[20];

void setup() {

size(200,200);
```

```
smooth();
// Inicjalizuj tablice
for (int i = 0; i < s.length; i + + ) {
s[i] = new Square(100,100,10);
}
for (int i = 0; i < c.length; i + + ) {
c[i] = new Circle(100,100,10,color(random(255),100));
}
}
void draw() {
background(100);
// Przełącz i wyświetl wszystkie kwadraty
for (int i = 0; i < s.length; i + + ) {
s[i].jiggle();
s[i].display();
}
// Przełącz i wyświetl wszystkie okręg
for (int i = 0; i < c.length; i + + ) {
c[i].jiggle();
c[i].display();
}
}
```

Polimorfizm pozwala nam uprościć powyższe, tworząc tylko jedną tablicę obiektów Shape, która zawiera zarówno obiekty Circle, jak i Square. Nie musimy się martwić o to, które z nich, wszystko to zostanie załatwione dla nas! (Zauważ też, że kod klas się nie zmienił, więc nie włączamy go tutaj.) Patrz przykład 22.2.

```
// Jedna tablica kształtów
Shape[] shapes = new Shape[30];        //Nowy sposób polimorfizmu z jedną tablicą, która ma różne
                                       //typy obiektów rozszerzających Shape.
void setup() {
size(200,200);
smooth();
```

```
for (int i = 0; i < shapes.length; i + + ) {
int r = int(random(2));
//  losowo wstaw kółka lub kwadraty do naszej tablicy
if (r = = 0) {
shapes[i] = new Circle(100,100,10,color(random(255),100));
} else {
shapes[i] = new Square(100,100,10);
}
}
}
void draw() {
background(100);
// Przełączaj i wyświetlaj wszystkie kształty
for (int i = 0; i < shapes.length; i + + ) {
shapes[i].jiggle();
shapes[i].display();
}
}
```

Ćwiczenie 22-4: Dodaj klasę linii utworzoną w ćwiczeniu 22-2 do przykładu 22-2. Losowo umieszczaj okręgi, kwadraty i linie w tablicy. Zauważ, że ledwo musisz zmienić dowolny kod (powinieneś jedynie edytować setup () powyżej).

Ćwiczenie 22-5: Zaimplementuj polimorfizm na szkicu wykonanym dla ćwiczenia 22-3.

22.5 Przeciążenie

W części 16 dowiedzieliśmy się, jak utworzyć obiekt przechwytywania w celu odczytu obrazów na żywo z kamery wideo. Jeśli spojrzałeś na stronę referencyjną Processing

(http://www.processing.org/reference/libraries/video/

Capture.html), możesz zauważyć, że konstruktor Capture może zostać wywołany z trzema, czterema lub pięcioma argumentami:

Capture(parent, width, height)

Capture(parent, width, height, fps)

Capture(parent, width, height, name)

Capture(parent, width, height, name, fps)

Funkcje, które mogą przyjmować różne liczby argumentów, są w rzeczywistości czymś, co widzieliśmy aż w części 1! Wypełnij (), na przykład, można wywołać z jednym numerem (dla skali szarości), trzema (dla koloru RGB) lub czterema (dla uwzględnienia przezroczystości alfa).

```
fill(255);

fill(255,0,255);

fill(0,0,255,150);
```

Możliwość definiowania funkcji o tej samej nazwie (ale różnych argumentach) jest nazywana przeciążeniem. Na przykład z fill (), Processing nie myli się co do definicji fi ll () do wyszukania, po prostu znajduje ten, w którym argumenty są zgodne. Nazwa funkcji w połączeniu z jej argumentami jest znana jako sygnatura funkcji - to właśnie czyni tę definicję funkcji wyjątkową. Przyjrzyjmy się przykładowi, który pokazuje, jak przydatne może być przeciążenie. Powiedzmy, że masz klasę Fish. Każdy obiekt Fish ma lokalizację: x i y.

```
class Fish {

float x;

float y;
```

Co zrobić, gdy tworzysz obiekt Fish, czasami chcesz zrobić obiekt z losową lokalizacją, a czasami z określoną lokalizacją. Aby było to możliwe, moglibyśmy napisać dwa różne konstruktory:

```
Fish() {

x = random(0,width);    //Przeciążenie pozwala nam zdefiniować dwa konstruktory dla tego samego
                        //obiektu (o ile te konstruktory przyjmują różne argumenty).

y = random(0,height);

}

Fish(float tempX, float tempY) {

x = tempX;

y = tempY;

}

}
```

Kiedy tworzysz obiekt Fish w głównym programie, możesz to zrobić za pomocą dowolnego konstruktora:

```
Fish fish1 = new Fish();

Fish fish2 = new Fish(100,200);
```

Jeśli zdefiniowano dwa konstruktory, obiekt można zainicjować za pomocą jednego z nich.

Java

23.1 Ujawnianie kreatora

Być może przez cały czas czytania myślałeś: „Boże, to jest takie niesamowite. Kocham programowanie. Cieszę się, że uczę się Processing. Nie mogę się doczekać, aby kontynuować i uczyć się języka Java! „Zabawne jest to, że tak naprawdę nie musisz uczyć się języka Java. Zdejmując zasłonę Processingu, odkryjemy, że cały czas uczymy się Java. Na przykład w Javie:

• Zadeklaruj, zainicjuj i używaj zmiennych w ten sam sposób.

• Zadeklaruj, zainicjuj i używaj tablic w ten sam sposób.

• Używaj instrukcji warunkowych i pętli w ten sam sposób.

• Definiowanie i wywoływanie funkcji w ten sam sposób.

• Twórz klasy w ten sam sposób.

• Uruchamianie obiektów w ten sam sposób.

Processing jest doskonałym narzędziem do nauki i tworzenia interaktywnych projektów graficznych. Oto lista niektórych funkcji przetwarzania, które nie są tak łatwo dostępne za pomocą zwykłej Java.

• Zestaw funkcji do rysowania kształtów.

• Zestaw funkcji do ładowania i wyświetlania tekstu, obrazów i wideo.

• Zestaw funkcji do transformacji 3D.

• Zestaw funkcji do obsługi myszy i klawiatury.

• Proste środowisko programistyczne do pisania kodu.

• Przyjazna społeczność internetowa artystów, projektantów i programistów!

23.2 Gdybyśmy nie mieli Processing, jak wyglądałby nasz kod?

W części 2 omówiliśmy proces kompilacji i wykonania - tak dzieje się po naciśnięciu przycisku odtwarzania i przekształceniu kodu w okno z grafiką. Krok 1 tego procesu polega na „przetłumaczeniu" kodu przetwarzania na język Java. W rzeczywistości podczas eksportowania do apletu następuje ten sam proces i zauważysz, że wraz z plikiem „Sketchname.pde" pojawi się nowy plik o nazwie „SketchName.java. „To jest Twój„ przetłumaczony "kod, tylko tłumaczenie jest nieco mylące, ponieważ niewiele zmian. Spójrzmy na przykład:

```
//  Losowo rosnący  kwadrat

float w = 30.0; // Variable to keep track of size of rect

void setup() {

size(200, 200);

}                              //To, do czego jesteśmy przyzwyczajeni, nasz zwykły stary kod przetwarzania.

void draw() {

background(100);
```

```
rectMode(CENTER);
fill(255);
noStroke();
rect(mouseX,mouseY,w,w); // Narysuj rect w miejscu myszy
w + = random( – 1,1); // Losowo dostosuj zmienną rozmiaru
}
```

Eksportując szkic, możemy otworzyć plik Java i spojrzeć na źródło Java.

```
// Losowo rosnący kwadrat z Java
import processing.core.*;
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.io.*;                    //Przetłumaczony kod Java ma kilka nowych rzeczy na górze,
                                     //ale wszystko inne pozostaje takie samo
import java.net.*;
import java.text.*;
import java.util.*;
import java.util.zip.*;
public class JavaExample extends PApplet {
float w = 30.0f; // Zmienna, aby śledzić rozmiar rect
public void setup() {
size(200, 200);
}
public void draw() {
background(100);
rectMode(CENTER);
fill(255);
noStroke();
rect(mouseX,mouseY,w,w); // narysuj rect w miejscu myszy
w + = random( – 1,1); //losowo dostosuj zmienną rozmiaru
```

}

}

Oczywiste jest, że niewiele się zmieniło, ale jakiś kod został dodany przed i po zwykłych ustawieniach setup () i draw ().

• Instrukcje dotyczące importu - u góry znajduje się zestaw instrukcji importowania umożliwiających dostęp do niektórych bibliotek. Widzieliśmy to już wcześniej, gdy korzystaliśmy z bibliotek Processing. Gdybyśmy używali zwykłej Javy zamiast Processing, zawsze wymagane byłoby podanie wszystkich bibliotek. Processing zakłada jednak podstawowy zestaw bibliotek z Java (np. Java.applet. *) I z Processing (np. Processing.core. *), które jest dlaczego nie widzimy ich w każdym szkicu.

• public - W Javie zmienne, funkcje i klasy mogą być „publiczne" lub „prywatne". "To oznaczenie wskazuje, jaki poziom dostępu powinien być przyznany konkretnemu fragmentowi kodu. Nie jest to coś, o co musimy się martwić w prostszym środowisku przetwarzania, ale staje się ono ważnym czynnikiem przy przechodzeniu do większych programów Java. Jako indywidualny programista najczęściej udzielasz lub odmawiasz dostępu do siebie, jako środka ochrony przed błędami. Spotkaliśmy się z kilkoma przykładami tej dyskusji w części 22 dotyczącej enkapsulacji.

• klasa JavaExample - dźwięk nieco znajomy? Okazuje się, że Java jest prawdziwym językiem obiektowym. Nie ma nic napisanego w Javie, który nie jest częścią klasy! Przyzwyczailiśmy się do idei klasy Zoog, klasy Car, klasy PImage itd., Ale ważne jest, aby pamiętać, że szkic jako całość jest również klasą! Przetwarzanie wypełnia te rzeczy dla nas, więc nie musimy się martwić o zajęcia, kiedy po raz pierwszy uczymy się programować.

• rozszerzone PApplet - Cóż, po przeczytaniu Części 22 powinniśmy być całkiem zadowoleni z tego, co to oznacza. To tylko kolejny przykład dziedziczenia. Tutaj klasa JavaExample jest dzieckiem klasy PApplet (lub, równoważnie, PApplet jest rodzicem JavaExample). PApplet jest klasą opracowaną przez twórców Processing i rozszerzając ją, nasz szkic ma dostęp do wszystkich elementów Processing goodies-setup (), draw (), mouseX, mouseY itd. Ten mały fragment kodu jest sekretem tego, jak prawie wszystko działa w szkicu przetwarzania.

Processing tak dobrze nam służyło, ponieważ eliminuje potrzebę martwienia się o powyższe cztery elementy, a jednocześnie zapewnia dostęp do zalet języka programowania Java. Pozostała część pokaże, jak możemy zacząć korzystać z dostępu do pełnego API Java

23.3 Eksploracja API Java

Referencje dotyczące przetwarzania szybko stały się naszym najlepszym przyjacielem na zawsze podczas nauki programowania. API Java zacznie się bardziej jako znajomy, na który wpadamy od czasu do czasu. Ten znajomy może pewnego dnia stać się naprawdę wspaniałym przyjacielem, ale na razie małe dawki będą po prostu drobne. Możemy zapoznać się z pełną dokumentacją Javy, odwiedzając:

http://java.sun.com/

Tam możemy kliknąć na specyfikacje API:

http://java.sun.com/reference/api/index.html

i znajdź wybór wersji Java. Na komputerach Mac Przetwarzanie będzie działać z wybraną wersją Java (uruchom Preferencje Java, znalezione np. W: / Applications / Utilities / Java / J2SE 5.0 /). Na PC, „Standardowy" Processing pochodzi z Java w wersji 1.4.2 (wersja ekspercka Windows pozwala zainstalować własną wersję Javy). Wersje Java zmienią się w przyszłości ze zaktualizowanymi

informacjami na processing.org. W każdym razie, podczas gdy istnieją różnice między wersją Javy, dla tego, co robimy, nie będą one bardzo istotne i możemy spojrzeć na API dla J2SE 1.4.2, aby uzyskać niemal wszystko, co chcemy zrobić.

I tak bardzo szybko zostaniesz całkowicie zagubiony. I to jest OK. Interfejs API Java jest ogromny. Nie jest tak naprawdę przeznaczone do czytania. To naprawdę czysta wzmianka o wyszukiwaniu określonych klas, o których wiesz, że musisz zajrzeć. Na przykład, możesz pracować nad programem, który wymaga wyrafinowanego generowania liczb losowych wykraczających poza to, co może zrobić losowy (), i podsłuchałeś rozmowę o klasie „Losowo" i pomyślałeś „Hej, może powinienem to sprawdzić! „Stronę referencyjną dla konkretnej klasy można znaleźć, przewijając listę„ Wszystkie klasy "lub wybierając odpowiedni pakiet (w tym przypadku pakiet java.util). Pakiet, podobnie jak biblioteka, jest zbiorem klas (API organizuje je według tematu). Najprostszym sposobem na znalezienie klasy jest wpisanie nazwy klasy i Java (np. „Java Random") do google. Strona dokumentacji jest prawie zawsze pierwszym wynikiem. Podobnie jak w przypadku odwołania do przetwarzania, strona Java zawiera wyjaśnienie tego, co robi klasa, konstruktory do tworzenia instancji obiektu oraz dostępne pola (zmienne) i metody (funkcje). Ponieważ Random jest częścią pakietu java.util (który przetwarza założenie importu), nie musimy jawnie pisać instrukcji importu, aby go użyć. Oto kod, który tworzy obiekt Random i wywołuje na nim funkcję nextBoolean (), aby uzyskać losową wartość true lub false.

Przykład 23-1: Używanie java.util.Random zamiast random ()

```
Random r;

void setup() {

size(200,200);

r = new Random();

}

void draw() {

boolean trueorfalse = r.nextBoolean();  //Wywołanie funkcji znalezionej pod adresem

//http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html

if (trueorfalse) {

background(0);

} else {

background(255);

}

}
```

Ćwiczenie 23-1: Odwiedź stronę referencyjną Java dla Random i użyj jej, aby uzyskać losową liczbę całkowitą z zakresu od 0 do 9. http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random .html.

23.4 Inne przydatne klasy Java: ArrayList

W części 6 dowiedzieliśmy się, jak używać tablicy do śledzenia uporządkowanych list informacji. Stworzyliśmy tablicę N obiektów i użyliśmy pętli „for", aby uzyskać dostęp do każdego elementu

tablicy. Rozmiar tej tablicy został naprawiony - ograniczyliśmy się do N i tylko N elementów. Istnieją alternatywy. Jedną z opcji jest użycie bardzo dużej tablicy i użycie zmiennej do śledzenia, ile macierzy użyć w danym momencie (patrz przykład łapacza deszczu w Rozdziale 10). Przetwarzanie również poza rozszerzeniami (), contract (), podzbiorem (), splice () i innymi metodami zmiany rozmiaru tablic. Byłoby jednak wspaniale mieć klasę, która implementuje tablicę o niezmiennej wielkości, umożliwiającą dodawanie lub usuwanie elementów z początku, środka i końca tej tablicy. Dokładnie to robi klasa Java ArrayList znajdująca się w pakiecie java.util. Strona referencyjna znajduje się tutaj: http://java.sun.com/j2se/1.4.2/docs/api/java/util/ArrayList.html.

Użycie ArrayList jest koncepcyjnie podobne do standardowej tablicy, ale składnia jest inna. Oto kod (który zakłada istnienie klasy „Particle") wykazujący identyczne wyniki, najpierw z tablicą, a drugi z ArrayList. Wszystkie metody użyte w tym przykładzie są udokumentowane na stronie odniesienia JavaDoc.

```
// STANDARDOWY SPOSÓB TABLIC

int MAX = 10;

//deklarowanie tablicy

Particle[] parray = new Particle[MAX];   //Standardowy sposób tablic. To właśnie robiliśmy od
                                         //początku, uzyskując dostęp do elementów tablicy za pomocą
                                         //indeksu i nawiasów kwadratowych— [].

// Zainicjuj tablicę w konfiguracji

void setup() {

for (int i = 0; i < parray.length; i + + ) {

parray[i] = new Particle();

}

}

// Pętla przez tablicę, aby wywołać metody w losowani

void draw() {

for (int i = 0; i < parray.length; i + + ) {

Particle p = parray[i];              // Nowy ArrayList Way

                                     //Elementy ArrayList są dodawane i dostępne za pomocą funkcji (a nie
                                     //nawiasów) —add () i get (). Wszystkie te funkcje są
                                     //udokumentowane w odnośniku Java:
        //http://java.sun.com/j2se/1.4.2/docs/ api / java / util / ArrayList.html.

p.run();

p.display();

}

}
```

```
// SPOSÓB NOWYCH ZMIAN ARRAYLIST

int MAX = 10;

// Deklarowanie i tworzenie instancji ArrayList

ArrayList plist = new ArrayList();

void setup() {

for (int i = 0; i < MAX; i + + ) {

plist.add(new Particle());          //Obiekt jest dodawany do tablicy ArrayList za pomocą add ()

}

}

void draw() {

for (int i = 0; i < plist.size(); i + + ) {        //Rozmiar tablicy ArrayList jest zwracany przez size().

Particle p = (Particle) plist.get(i);

p.run();                    //Dostęp do obiektu uzyskuje się z ArrayList za pomocą get (). Musisz
                            //„rzutować” cokolwiek wychodzi z ArrayList na typ, to znaczy (Cząsteczka).

p.display();

}

}
```

W tej ostatniej pętli „for” musimy rzucić obiekt, gdy wyciągniemy go z listy ArrayList. Lista ArrayList nie śledzi typu przechowywanych obiektów - nawet jeśli może wydawać się zbędna, naszym zadaniem jest przypominanie programowi poprzez umieszczenie typu klasy (tj. Cząsteczka) w nawiasach. Ten proces jest znany jako casting. Oczywiście powyższy przykład jest nieco głupi, ponieważ nie wykorzystuje możliwości zmiany rozmiaru ArrayList, używając ustalonego rozmiaru 10. Poniżej podano lepszy przykład, który dodaje jedną nową cząsteczkę do tablicy ArrayList z każdy cykl poprzez draw (). Upewniamy się również, że ArrayList nigdy nie otrzyma więcej niż 100 cząstek.

Termin „system cząstek” został ukuty w 1983 roku przez Williama T. Reevesa, który opracował efekt „Genesis” na koniec filmu „Star Trek II: The Wrath of Khan”. System cząstek jest zazwyczaj zbiorem niezależnych obiektów, zwykle przedstawianych za pomocą prostego kształtu lub kropki. Może być używany do modelowania typów zjawisk naturalnych, takich jak eksplozje, ogień, dym, iskry, wodospady, chmury, mgła, płatki, trawa, bąbelki i tak dalej.

Przykład 23-2: Prosty system cząstek z ArrayList

```
ArrayList particles;

void setup() {

size(200,200);

particles = new ArrayList();
```

```
smooth();
}
void draw() {
particles.add(new Particle());    //Nowy obiekt Particle jest dodawany do ArrayList w każdym cyklu
                                  //poprzez draw ().
background(255);
//  Iteruj przez naszą tablicę ArrayList i pobierz każdą cząstkę
for (int i = 0; i < particles.size(); i + + ) {
Particle p = (Particle) particles.get(i);    //ArrayList śledzi, ile elementów jest przechowywanych, a
                                             //następnie przeglądamy je wszystkie.
p.run();
p.gravity();
p.display();
}
// Usuń pierwszą cząstkę, gdy lista przekroczy 100.
if (particles.size() > 100) {
particles.remove(0);    //Jeśli ArrayList zawiera więcej niż 100 elementów, usuwamy pierwszy
                        //element, używając remove().
}
}
// Prosta klasa cząstek
class Particle {
float x;
float y;
float xspeed;
float yspeed;
Particle() {
x = mouseX;
y = mouseY;
xspeed = random(–1,1);
yspeed = random(–2,0); }
void run() {
```

x = x + xspeed;

y = y + yspeed;

}

void gravity() {

yspeed + = 0.1;

}

void display() {

stroke(0);

fill(0,75);

ellipse(x,y,10,10);

}

}

Ćwiczenie 23-2: Przepisz przykład 23-2, aby cząstki były usuwane z listy za każdym razem, gdy opuszczą okno (tj. Ich położenie y jest większe niż wysokość).

Podpowiedź: Dodaj funkcję, która zwraca wartość logiczną w klasie cząstek.

_____ skończone() {

Jeśli (_____) {

powrót _____;

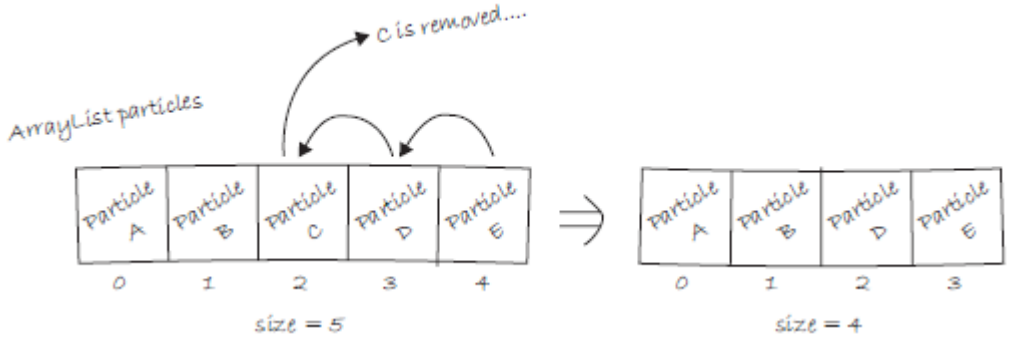
} else {

return false;

}

}

Podpowiedź: Aby to działało poprawnie, musisz przechodzić przez elementy ArrayList do tyłu! Czemu? Ponieważ po usunięciu elementu wszystkie kolejne elementy są przesuwane w lewo



for (int i = _____; i _____; i_____) {

```
Particle p = (Particle) particles.get(i);

p.run();

p.gravity();

p.render();

if (_______) {

_______;

}

}
```

23.5 Inne przydatne klasy Java: Prostokąt

Druga pomocna klasa Java, którą zbadamy, to klasa Rectangle:

http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Rectangle.html.

Prostokąt Java określa obszar w przestrzeni współrzędnych, który jest otoczony przez lewy górny punkt obiektu Rectangle (x, y), jego szerokość i wysokość. Brzmi znajomo? Oczywiście funkcja przetwarzania rect () rysuje prostokąt o dokładnie takich samych parametrach. Klasa e Rectangle hermetyzuje ideę prostokąta w obiekcie. Prostokąt Java ma przydatne metody, na przykład zawiera (). zawiera () umożliwia prosty sposób sprawdzenia, czy punkt lub prostokąt znajduje się wewnątrz tego prostokąta, poprzez otrzymanie współrzędnej xiy i zwrócenie wartości true lub false, na podstawie tego, czy punkt znajduje się wewnątrz prostokąta, czy nie. Oto prosty najazd zaimplementowany za pomocą obiektu Rectangle i cntains(). Patrz przykład 23.3

Przykład 23-3: Używanie obiektu java.awt.Rectangle

```
Rectangle rect1, rect2;

void setup() {

size(200,200);

rect1 = new Rectangle(25,25,50,50);     //Ten szkic wykorzystuje dwa obiekty Rectangle. Argumenty
                                        //konstruktora (x, y, szerokość, wysokość) są
                                        //udokumentowane w odwołaniu do Java:

// http: //java.sun. pl / j2se / 1.4.2 / docs / api / java / awt / Recangle.html.

rect2 = new Rectangle(125,75,50,75);

}

void draw() {

background(255);

stroke(0);

if (rect1.contains(mouseX,mouseY)) {    //Funkcja include() służy do określenia, czy mysz znajduje się
                                        //wewnątrz prostokąta.

fill(200);
```

```
} else {

fill(100);

}

rect(rect1.x, rect1.y, rect1.width,rect1.height); //Obiekt Rectangle wie tylko o zmiennych związanych
                                                  //z prostokątem. Nie może go wyświetlić. Używamy
                                                  //więc funkcji rect () Przetwarzania w połączeniu z
                                                  //danymi Prostokąta.

// Repeat for the second Rectangle

// (of course, we could use an array or ArrayList here!)

if (rect2.contains(mouseX,mouseY)) {

fill(200);

} else {

fill(100);

}

rect(rect2.x, rect2.y, rect2.width,rect2.height);

}
```

Zabawmy się i połączmy przykład „cząstki" ArrayList z najazdem „Prostokąt". "W przykładzie 23-4 cząstki są wytwarzane w każdej ramie i przyciągane grawitacyjnie w dół okna. Jeśli jednak trafią na prostokąt, zostają złapani. Cząstki są przechowywane w tablicy ArrayList, a obiekt Rectangle określa, czy zostały one złapane, czy nie. (Ten przykład zawiera odpowiedzi na ćwiczenie 23-2.)

Przykład 23-4: Super fantazyjny system tablic i prostokątów

```
// Deklaracja zmiennej globalnej typu ArrayList

ArrayList particles;

// „Prostokąt" wysysa cząstki

Rectangle blackhole;

void setup() {

size(200,200);

blackhole = new Rectangle(50,150,100,25);

particles = new ArrayList();

smooth();

}

void draw() {

background(255);
```

```
// Wyświetlanie prostokąta
stroke(0);
fill(175);
rect(blackhole.x, blackhole.y, blackhole.width,blackhole.height);
// Dodaj nową cząstkę w miejscu myszy
particles.add(new Particle(mouseX,mouseY));
// Pętla przez wszystkie cząstki
for (int i = particles.size() – 1; i > = 0; i – – ) {
Particle p = (Particle) particles.get(i);
p.run();
p.gravity();
p.display();
if (blackhole.contains(p.x,p.y)) {//Jeśli prostokąt zawiera położenie cząstki, zatrzymaj cząstkę przed
                                  //poruszeniem się.
p.stop();
}
if (p.finished()) {
particles.remove(i);
}
}
}
// Prosta klasa cząstek
class Particle {
float x;
float y;
float xspeed;
float yspeed;
float life;
// Zrób cząstkę
Particle(float tempX, float tempY) {
x = tempX;
```

```
y = tempY;
xspeed = random(–1,1);
yspeed = random(–2,0);
life = 255;
}
// Ruszaj się
void run() {
x = x + xspeed;
y = y + yspeed;
}
// Upaść
void gravity() {
yspeed + = 0.1;
}
// Przestań się ruszać
void stop() {
xspeed = 0;
yspeed = 0;
}
// Gotowy do usunięcia
boolean finished() {
life – = 2.0;                    //Cząstka ma zmienną „życie”, która maleje. Kiedy osiągnie 0, cząstka
                                 //może zostać usunięta z listy ArrayList.
if (life < 0) return true;
else return false;
}
//PokaŻ
void display() {
stroke(0);
fill(0,life);
ellipse(x,y,10,10);
```

}

}

Ćwiczenie 23-3: Napisz klasę Button zawierającą obiekt Rectangle, aby określić, czy mysz kliknęła w obszarze przycisku. (To jest rozszerzenie ćwiczenia 9-8.)

23.6 Obsługa wyjątków (błąd)

Oprócz bardzo dużej biblioteki przydatnych klas, język programowania Java zawiera pewne funkcje wykraczające poza to, co omówiliśmy podczas nauki przetwarzania. Przeanalizujemy teraz jedną z tych funkcji: obsługa wyjątków. Błędy popełniane podczas programowania. Wszyscy je widzieliśmy.

java.lang.ArrayIndexOutOfBoundsException

java.io.IOException: openStream () nie mógł otworzyć pliku.jpg

java.lang.NullPointerException

To smutne, gdy te błędy występują. Naprawdę tak jest. Komunikat o błędzie jest drukowany i program rozprasza się, nigdy nie kontynuuje pracy. Być może opracowałeś kilka technik ochrony przed tymi komunikatami o błędach. Na przykład:

if (index <somearray.length) {

somearray [index] = random (0,100);

}

Powyższe jest formą „sprawdzania błędów. ”Kod używa warunku, aby sprawdzić, czy indeks jest prawidłowy, zanim dojdzie do dostępu do elementu w tym indeksie. Powyższy kod jest dość sumienny i wszyscy powinniśmy być ostrożni. Jednak nie wszystkie sytuacje są tak proste, aby uniknąć, i to jest, gdy w grę wchodzi obsługa wyjątków. Obsługa wyjątków odnosi się do procesu obsługi wyjątków, poza zwykłymi zdarzeniami (tj. Błędami), które występują podczas wykonywania programu. Struktura kodu do obsługi wyjątków w Javie jest znana jako try catch. Innymi słowy, „Spróbuj uruchomić jakiś kod. Jeśli napotkasz problem, złap błąd i uruchom inny kod. ”Jeśli zostanie wykryty błąd przy użyciu try catch, program może kontynuować. Przyjrzyjmy się, jak możemy przepisać kod sprawdzający błąd indeksu tablicy, spróbuj stylu catch.

try {

somearray[index] = 200;

} catch (Exception e) { //Kod, który może powodować błąd, znajduje się w sekcji „try”. Kod, który
                         //powinien się zdarzyć, jeśli wystąpi błąd, znajduje się w sekcji „catch”.

println("Hey, that’s not a valid index! ");

}

Powyższy kod przechwytuje każdy możliwy wyjątek, który może wystąpić. Rodzaj złapanego błędu to ogólny wyjątek. Jeśli jednak chcesz wykonać określony kod na podstawie określonego wyjątku, możesz, jak pokazuje poniższy kod. try {

somearray[index] = 200;

} catch (ArrayIndexOutOfBoundsException e) {

```
println ( "Hey, that's not a valid index! ");

} catch (NullPointerException e) {          //Różne sekcje „catch" wychwytują różne rodzaje
                                            //wyjątków. Ponadto każdy wyjątek jest obiektem i
                                            //dlatego może mieć metody . Na przykład:

                                            //e.printStackTrace ();

                                            //wyświetla bardziej szczegółowe informacje o
                                            //wyjątku.

println( "I think you forgot to create the array! ");

} catch (Exception e) {

println( "Hmmm, I dunno, something weird happened! ");

e.printStackTrace();

}
```

Powyższy kod nie robi jednak nic poza drukowaniem niestandardowych komunikatów o błędach. Są sytuacje, w których chcemy więcej niż tylko wyjaśnienia. Weźmy na przykład przykłady z części 18, w której załadowaliśmy dane ze ścieżki URL. Co jeśli nasz szkic nie zdołał połączyć się z adresem URL? Rozbiłby się i odszedł. Dzięki obsłudze wyjątków możemy przechwycić ten błąd i ręcznie uzupełnić tablicę ciągami, aby szkic mógł być kontynuowany.

```
String[] lines;

String url = " http://wwww.rockinurl.com " ;

try {

lines = loadStrings(url);

} catch (Exception e) {

lines = new String[3];

lines[0] = "Nie mogłem połączyć się z" + url + "!!! " ;     //Jeśli wystąpi problem z połączeniem z
                                                            //adresem URL, wypełnimy tablica z
                                                            //obojętnym tekstem, aby szkic mógł nadal
                                                            //działać.

lines[1] = "Ale tutaj jest kilka rzeczy do pracy " ;

lines[2] = " tak czy inaczej. " ;

}

println(lines);
```

23.7 Java poza przetwarzaniem

Oto ostatnia sekcja. Jeśli chcesz, możesz teraz zrobić sobie przerwę. Może wyjdź na zewnątrz i idź na spacer. Nawet trochę pobiegać. To będzie dla ciebie dobre. OK, wróć teraz? Świetnie, chodźmy dalej. Kiedy zamykamy ten rozdział i książkę, omówimy jeden ostatni temat: co zrobić, jeśli i kiedy zdecydujesz, że chcesz rozpocząć kodowanie poza przetwarzaniem. Ale dlaczego kiedykolwiek

chciałbym to zrobić? Jedna instancja, która zasługiwałaby na inne środowisko programistyczne, polega na stworzeniu aplikacji, która nie zawiera żadnej grafiki! Być może trzeba napisać program, który pobiera wszystkie informacje finansowe z arkuszy kalkulacyjnych i rejestruje je w bazie danych. Lub serwer czatu działający w tle na komputerze. Chociaż oba z nich mogą być zapisane w środowisku przetwarzania, niekoniecznie potrzebują żadnej funkcji przetwarzania do uruchomienia. W przypadku tworzenia większych i większych projektów, które obejmują wiele klas, środowisko przetwarzania może stać się nieco trudne w użyciu. Na przykład, jeśli twój projekt obejmuje, powiedzmy, 20 klas. Tworzenie szkicu przetwarzania z 20 kartami (a jeśli to było 40? Lub 100?) Może być trudne do zarządzania, nie mówiąc już o dopasowaniu do ekranu. W tym przypadku lepsze byłoby środowisko programistyczne zaprojektowane dla projektów Java na dużą skalę. Ponieważ Przetwarzanie to Java, nadal można korzystać ze wszystkich funkcji dostępnych w Przetwarzaniu w innych środowiskach programistycznych, importując biblioteki podstawowe. Więc co robisz? Po pierwsze, powiedziałbym, że nie spiesz się. Ciesz się przetwarzaniem i poświęć trochę czasu, aby poczuć się komfortowo dzięki własnemu procesowi kodowania i stylowi, zanim zaczniesz się interesować niezliczonymi problemami i pytaniami, które pojawiają się wraz z programowaniem w Javie. Jeśli czujesz się gotowy, pierwszym krokiem jest po prostu poświęcenie czasu na przeglądanie strony internetowej Java:

http://java.sun.com/

Zacznij od jednego z samouczków:

http://java.sun.com/docs/books/tutorial/

Samouczki obejmą ten sam materiał znaleziony w tej książce, ale z czystej perspektywy Java. Następnym krokiem byłoby wypróbowanie kompilacji i uruchomienia programu Java „Hello World”.

```
public class HelloWorld {

public static void main(String[] args) {

System.out.println( "Hello World. I miss you Processing. ");

}

}
```

Witryna Java zawiera samouczki, które wyjaśniają wszystkie elementy programu Hello World, a także zawierają instrukcje dotyczące kompilowania i uruchamiania go w wierszu poleceń. ”Http://java.sun.com/docs/books/tutorial/getStarted/index.html

Wreszcie, jeśli czujesz się ambitny, idź i pobierz Eclipse:

http://www.eclipse.org/

Eclipse to środowisko programistyczne dla Java z wieloma zaawansowanymi funkcjami. Niektóre z tych funkcji sprawią, że boli cię głowa, a niektóre sprawią, że będziesz się zastanawiać, jak zaprogramowałeś bez Eclipse. Pamiętaj, że po rozpoczęciu przetwarzania w rozdziale 2 prawdopodobnie uruchomiłeś pierwszy szkic w mniej niż pięć minut. Z czymś takim jak Eclipse, możesz potrzebować dużo więcej czasu, aby zacząć. Odwiedź stronę internetową tej książki, aby uzyskać linki i samouczki na temat uruchamiania szkicu przetwarzania w środowisku Eclipse.