Klasy, struktury i wyliczenia

W poprzedniej części nauczyłeś się grupować sekwencje instrukcji za pomocą funkcji i zamknięć. Czas pomyśleć o tym, jak reprezentować złożone obiekty w swoim kodzie. Pomyśl na przykład o samochodzie. Możesz użyć stałej String do przechowywania nazwy samochodu i zmiennej Double do przechowywania ceny samochodu, ale nie są one ze sobą powiązane. Widziałeś, że możesz grupować instrukcje, aby tworzyć funkcje i zamknięcia. W tym rozdziale dowiesz się, jak grupować stałe i zmienne w jedną całość przy użyciu klas i struktur oraz jak nimi manipulować. Dowiesz się również, jak używać wyliczeń, aby zgrupuować zestaw powiązanych wartości. Pod koniec tego rozdziału dowiesz się, jak tworzyć i inicjować klasę, tworzyć podklasę z istniejącej klasy, tworzyć i inicjować strukturę, rozróżniać klasy i struktury oraz tworzyć wyliczenie. Tu zostaną omówione następujące tematy:

• Zrozumienie klas

• Zrozumienie struktur

• Zrozumienie wyliczeń

Zrozumienie klas

Klasy są przydatne do reprezentowania złożonych obiektów, na przykład:

• Indywidualne informacje o pracownikach dla firmy

• Przedmioty na sprzedaż w witrynie e-commerce

• Przedmioty, które masz w domu do celów ubezpieczeniowych

Oto jak wygląda deklaracja i definicja klasy:

```
class ClassName {

property1

property2

property3

method1() {

code

}

method2() {

code

}

}
```

Każda klasa ma opisową nazwę i zawiera zmienne lub stałe używane do reprezentowania obiektu. Zmienne lub stałe skojarzone z klasą nazywane są właściwościami. Klasa może również zawierać funkcje, które wykonują określone zadania. Funkcje związane z klasą nazywane są metodami. Po zadeklarowaniu i zdefiniowaniu klasy możesz tworzyć instancje tej klasy. Wyobrażać sobie ,że tworzysz aplikację dla zoo. Jeśli masz klasę Animal, możesz użyć instancji tej klasy do reprezentowania różnych typów zwierząt w zoo. Każde z tych wystąpień będzie miało inne wartości dla swoich właściwości.

Zobaczmy, jak pracować z klasami. Dowiesz się, jak deklarować i definiować klasy, tworzyć instancje na podstawie deklaracji klasy i manipulować tymi instancjami. Zaczniesz od utworzenia deklaracji klasy reprezentującej zwierzęta w następnej sekcji.

Tworzenie deklaracji klasy

Zadeklarujmy i zdefiniujmy klasę, która może przechowywać szczegóły o zwierzętach. Dodaj następujący kod do swojego placu zabaw:

```
class Animal {

var name: String = ""

var sound: String = ""

var numberOfLegs: Int = 0

var breathesOxygen: Bool = true

func makeSound() {

print(self.sound)

}

}
```

Właśnie zadeklarowałeś bardzo prostą klasę o nazwie Animal. Konwencja nakazuje, aby nazwy klas zaczynały się od dużej litery. Ta klasa ma właściwości przechowywania imienia zwierzęcia, wydawanego przez nie dźwięku, liczby jego nóg oraz tego, czy oddycha ono tlenem, czy nie. Ta klasa ma również metodę makeSound(), która drukuje hałas, jaki wytwarza w obszarze Debug. Teraz, gdy masz już klasę Animal, użyjmy jej do utworzenia instancji zwierzęcia w następnej sekcji.

Tworzenie instancji klasy

Po zadeklarowaniu i zdefiniowaniu klasy możesz tworzyć instancje tej klasy. Utworzysz teraz instancję klasy Animal, która reprezentuje kota. Wykonaj następujące kroki:

1. Aby utworzyć instancję klasy Animal, wypisz wszystkie jej właściwości i wywołaj metodę makeSound(), wpisz po deklaracji klasy i uruchom ją:

```
let cat = Animal()

print(cat.name)

print(cat.sound)

print(cat.numberOfLegs)

print(cat.breathesOxygen)

cat.makeSound()
```

Dostęp do właściwości i metod instancji można uzyskać, wpisując kropkę po nazwie instancji, a następnie odpowiednią właściwość lub metodę. Zobaczysz wartości właściwości wystąpienia i wywołań metod wymienionych w obszarze debugowania. Ponieważ wartości są wartościami domyślnymi przypisywanymi podczas tworzenia klasy, nazwa i dźwięk zawierają puste ciągi,

numberOfLegs zawiera 0, breathesOxygen zawiera wartość true, a metoda makeSound() drukuje pusty ciąg.

2. Przypiszmy pewne wartości do właściwości tej instancji. Zmodyfikuj swój kod, jak pokazano:

```
let cat = Animal()

cat.name = "Cat"

cat.sound = "Mew"

cat.numberOfLegs = 4

cat.breathesOxygen = true

print(cat.name)
```

Teraz, po uruchomieniu programu, w obszarze Debug wyświetlane są następujące informacje:

```
Cat

Mew

4

true

Mew
```

Wartości wszystkich właściwości instancji i wynik metody makeSound() są wyświetlane w obszarze Debug. Zauważ, że tutaj najpierw tworzysz instancję, a następnie przypisujesz jej wartości. Możliwe jest również przypisanie wartości podczas tworzenia instancji, a robisz to poprzez zaimplementowanie inicjatora w deklaracji klasy.

3. Inicjator jest odpowiedzialny za zapewnienie właściwości całej instancji aby miały prawidłowe wartości podczas tworzenia klasy. Dodajmy inicjator dla klasy Animal. Zmodyfikuj definicję klasy, jak pokazano:

```
class Animal {

var name: String

var sound: String

var numberOfLegs: Int

var breathesOxygen: Bool

init(name: String, sound: String, numberOfLegs:

Int, breathesOxygen: Bool) {

self.name = name

self.sound = sound

self.numberOfLegs = numberOfLegs
```

```
self.breathesOxygen = breathesOxygen

}

func makeSound() {

print(self.sound)

}

}
```

Jak widać, inicjator używa słowa kluczowego init i ma listę parametrów, które zostaną użyte do ustawienia wartości właściwości. Należy zauważyć, że słowo kluczowe self odróżnia nazwy właściwości od parametrów. Na przykład ja. nazwa odnosi się do właściwości, a nazwa odnosi się do parametru. Pod koniec procesu inicjalizacji każda właściwość w klasie powinna mieć poprawną wartość.

4. W tym momencie zobaczysz kilka błędów w swoim kodzie. Aby rozwiązać ten problem, musisz zaktualizować wywołanie funkcji. Zmodyfikuj swój kod tak, jak pokazano i uruchom go:

```
func makeSound() {

print(self.sound)

}

}

let cat = Animal(name: "Cat", sound: "Mew",


numberOfLegs: 4, breathesOxygen: true)

print(cat.name)
```

Wyniki są takie same jak w kroku 2, ale utworzyłeś instancję i ustawiłeś jej właściwości w jednej instrukcji. Doskonale!

Obecnie istnieją różne rodzaje zwierząt, takie jak ssaki, ptaki, gady i ryby. Możesz utworzyć klasę dla każdego typu, ale możesz również utworzyć podklasę opartą na istniejącej klasie. Zobaczmy, jak to zrobić w następnej sekcji.

Tworzenie podklasy

Podklasa klasy dziedziczy wszystkie metody i właściwości istniejącej klasy. Jeśli chcesz, możesz również dodać do niego dodatkowe właściwości i metody. Utworzysz teraz Mammal, podklasę klasy Animal. Wykonaj następujące kroki:

1. Aby zadeklarować klasę Mammal wpisz następujący kod zaraz po deklaracji klasy Animal:

```
class Mammal: Animal {

let hasFurOrHair: Bool = true

}
```

Typing : Animal po nazwie klasy sprawia, że klasa Mammal staje się podklasą klasy Animal. Posiada wszystkie właściwości i metody zadeklarowane w klasie Animal oraz jedną dodatkową właściwość

hasFurOrHair. Ponieważ klasa Animal jest rodzicem klasy Mammal, można odnosić się do niej jako do klasy nadrzędnej klasy Mammal.

2. Zmodyfikuj swój kod, który tworzy instancję Twojej klasy, tak jak pokazano, i uruchom go:

let cat = Mammal(name: "Cat", sound: "Mew",

numberOfLegs: 4, breathesOxygen: true)

cat jest teraz instancją klasy Mammal zamiast klasy Animal. Jak widać, wyniki wyświetlane w obszarze Debug są takie same jak poprzednio i nie ma błędów. Wartość hasFurOrHair nie została jednak wyświetlona. Naprawmy to.

3. Wpisz następujący kod po wszystkich innych kodach na twoim placu zabaw, aby wyświetlić zawartość właściwości hasFurOrHair i uruchom ją:

print(cat.maFurOrHair)

Ponieważ inicjator klasy Animal nie ma parametru do przypisania wartości do hasFurOrHair, używana jest wartość domyślna, a true zostanie wyświetlona w obszarze Debug. Widziałeś, że podklasa może mieć dodatkowe właściwości. Podklasa może mieć również dodatkowe metody, a implementacja metody w podklasie może się różnić

Zastępowanie metody nadklasy

Do tej pory używałeś wielu instrukcji print() do wyświetlania wartości instancji klasy. Zaimplementujesz metodę description(), aby wyświetlić wszystkie właściwości instancji w obszarze Debug, dzięki czemu wielokrotne instrukcje print() nie będą już wymagane. Wykonaj następujące kroki:

1. Zmodyfikuj deklarację klasy Animal, aby zaimplementować metodę description(), jak pokazano:

class Animal {

var name: String

var sound: String

var numberOfLegs: Int

var breathesOxygen: Bool = true

init(name: String, sound: String, numberOfLegs:

Int, breathesOxygen: Bool) {

self.name = name

self.sound = sound

self.numberOfLegs = numberOfLegs

self.breathesOxygen = breathesOxygen

}

func makeSound() {

print(self.sound)

```
}

func description() -> String {

return "name: \(self.name)

sound: \(self.sound)

numberOfLegs: \(self.numberOfLegs)

breathesOxygen: \(self.breathesOxygen)"

}

}
```

2. Zmodyfikuj swój kod, jak pokazano, aby użyć metody description() zamiast wielu instrukcji print() i uruchom program:

```
let cat = Mammal(name: "Cat", sound: "Mew",

numberOfLegs: 4, breathesOxygen: true)

print(cat.description())

cat.makeSound()
```

W obszarze Debug zobaczysz następujące elementy:

```
name:  Cat sound: Mew numberOfLegs: 4 breathesOxygen: true

Mew
```

Jak widać, mimo że metoda description() nie jest zaimplementowana w klasie Mammal, jest zaimplementowana w klasie Animal. Oznacza to, że zostanie on odziedziczony przez klasę Mammal, a właściwości instancji zostaną wydrukowane w obszarze Debug. Należy zauważyć, że brakuje wartości właściwości hasFurOrHair i nie można jej umieścić w metodzie description(), ponieważ właściwość hasFurOrHair nie istnieje dla klasy Animal.

3. Możesz zmienić implementację metody description() w klasie Mammal, aby wyświetlić wartość właściwości hasFurOrHair. Dodaj następujący kod do definicji klasy Mammal i uruchom go:

```
Mammal: Animal {

let hasFurOrHair: Bool = true

override func description() -> String {

return super.description() + " hasFurOrHair:

\(self.hasFurOrHair)"

}

}
```

Słowo kluczowe override jest tutaj używane do określenia, że zaimplementowana tutaj metoda description() ma zostać użyta zamiast implementacji nadklasy. Słowo kluczowe super służy do

wywołania implementacji superklasy description(). Wartość w hasFurOrHair jest następnie dodawana do łańcucha zwracanego przez super. description().

W obszarze Debug zobaczysz następujące elementy:

name: Cat sound: Mew numberOfLegs: 4 breathesOxygen: true

hasFurOrHair: true

Mew

Wartość właściwości hasFurOrHair jest wyświetlana w obszarze Debug, co wskazuje, że używasz implementacji podklasy Mammal metody description(). Utworzyłeś deklaracje klas i podklas oraz stworzyłeś instancje obu. Do obu dodano także inicjatory i metody. Fajny! Przyjrzyjmy się, jak deklarować i używać struktur w następnej sekcji.

Zrozumienie struktur

Podobnie jak klasy, struktury również grupują właściwości i metody używane do reprezentowania obiektu i wykonywania określonych zadań. Pamiętasz klasę Animal, którą utworzyłeś? Możesz również użyć struktury, aby osiągnąć to samo. Istnieją jednak różnice między klasami i strukturami, o których dowiesz się później. Oto jak wygląda deklaracja i definicja struktury:

struct StructName {

property1

property2

property3

method1() {

code

}

method2(){

code

}

}

Jak widać, struktura jest bardzo podobna do klasy. Ma również opisową nazwę, może zawierać właściwości i metody oraz umożliwia tworzenie instancji. Przyjrzyjmy się, jak pracować ze strukturami. Dowiesz się, jak deklarować i definiować struktury, tworzyć instancje w oparciu o strukturę i manipulować nimi. Zaczniesz od stworzenia struktury reprezentującej gady w następnej sekcji.

Tworzenie deklaracji struktury

Kontynuując motyw zwierzęcy, zadeklarujmy i zdefiniujmy strukturę, która może przechowywać szczegóły gadów. Dodaj następujący kod po wszystkich innych kodach na swoim placu zabaw:

struct Reptile {

var name: String

```
var sound: String

var numberOfLegs: Int

var breathesOxygen: Bool

let hasFurOrHair: Bool = false

func makeSound() {

print(sound)

}

func description() -> String {

return "Structure: Reptile name: \(self.name)

sound: \(self.sound)

numberOfLegs: \(self.numberOfLegs)

breathesOxygen: \(self.breathesOxygen)

hasFurOrHair: \(self.hasFurOrHair)"

}

}
```

Jak widać, jest to prawie to samo, co deklaracja klasy Animal, którą zrobiłeś wcześniej. Nazwy struktur również zwykle zaczynają się wielką literą, a ta struktura ma właściwości przechowywania nazwy zwierzęcia, wydawanego przez nie dźwięku, liczby nóg, tego, czy oddycha tlenem i czy ma futro czy sierść. Ta struktura ma również metodę makeSound(), która drukuje dźwięk, który tworzy w obszarze Debug. Teraz, gdy masz już deklarację struktury Reptile, użyjmy jej do utworzenia instancji reprezentującej węża w następnej sekcji.

Tworzenie instancji struktury

Podobnie jak w przypadku klas, możesz tworzyć instancje z deklaracji struktury. Utworzysz teraz instancję struktury Reptile, która reprezentuje węża, wydrukujesz wartości właściwości tej instancji i wywołasz metodę makeSound(). Wpisz następujące polecenie po deklaracji struktury i uruchom ją:

```
var snake = Reptile(name: "Snake", sound: "Hiss",

numberOfLegs: 0, breathesOxygen: true)

print(snake.description())

snake.makeSound()
```

Zauważ, że nie musiałeś implementować inicjatora; struktury automatycznie uzyskują inicjator dla wszystkich swoich właściwości, zwany inicjatorem elementów członkowskich. Schludny! W obszarze debugowania zostaną wyświetlone:

```
Structure: Reptile name: Snake sound: Hiss numberOfLegs: 0

breathesOxygen: true hasFurOrHair: false
```

Hiss

Mimo że deklaracja struktury jest bardzo podobna do deklaracji klasy, istnieją dwie różnice między klasą a strukturą:

• Struktury nie mogą dziedziczyć z innej struktury.

• Klasy są typami referencyjnymi, podczas gdy struktury są typami wartościowymi.

Przyjrzyjmy się różnicy między typami wartości a typami odwołań w następnej sekcji.

Porównanie typów wartości i typów referencyjnych

Klasy są typami referencyjnymi. Oznacza to, że kiedy przypisujesz instancję klasy do zmiennej, w rzeczywistości przechowujesz lokalizację pamięci oryginalnej instancji w zmiennej, a nie samą instancję. Struktury to typy wartości. Oznacza to, że gdy przypiszesz instancję struktury do zmiennej, instancja ta zostanie skopiowana, a wszelkie zmiany wprowadzone do oryginalnej instancji nie mają wpływu na kopię. Teraz utworzysz instancję klasy i struktury oraz zaobserwujesz różnice między nimi. Wykonaj następujące kroki:

1. Zaczniesz od utworzenia zmiennej zawierającej instancję struktury i przypisania jej do drugiej zmiennej, a następnie zmienisz wartość właściwości w drugiej zmiennej. Wpisz następujący kod i uruchom go:

```
struct SampleValueType {

var sampleProperty = 10

}

var a = SampleValueType()

var b = a

b.sampleProperty = 20

print(a.sampleProperty)

print(b.sampleProperty)
```

W tym przykładzie zadeklarowałeś strukturę SampleValueType, która zawiera jedną właściwość sampleProperty. Następnie utworzyłeś instancję tej struktury i przypisałeś ją do zmiennej, a. Następnie przypisałeś a do nowej zmiennej, b. Następnie zmieniłeś wartość sampleProperty z b na 20. Po wydrukowaniu wartości sampleProperty a, 10 jest drukowane w obszarze Debug, pokazując , że wszelkie zmiany wprowadzone w wartości sampleProperty w b nie wpływają na wartość sampleProperty w a. Dzieje się tak, ponieważ po przypisaniu a do b kopia a została przypisana do b, więc są to całkowicie oddzielne instancje, które nie wpływają na siebie nawzajem.

2. Następnie utworzysz zmienną zawierającą instancję klasy i przypiszesz ją do drugiej zmiennej, a następnie zmienisz wartość właściwości w drugiej zmiennej. Wpisz następujący kod i uruchom go:

```
class SampleReferenceType {

var sampleProperty = 10

}
```

```
var c = SampleReferenceType()

var d = c

c.sampleProperty = 20

print(c.sampleProperty)

print(d.sampleProperty)
```

W tym przykładzie zadeklarowałeś klasę SampleReferenceType, która zawiera jedną właściwość sampleProperty. Następnie utworzyłeś instancję tej klasy i przypisałeś ją do zmiennej, c. Następnie przypisałeś c do nowej zmiennej, d. Następnie zmieniłeś wartość sampleProperty d na 20. Podczas drukowania wartości sampleProperty c, 20 jest drukowane w obszarze Debug, pokazując, że wszelkie zmiany wprowadzone w c lub d wpływają na tę samą instancję SampleReferenceType. Teraz pytanie brzmi, jakich klas czy struktur użyć? Przyjrzyjmy się temu w następnej sekcji.

Decydowanie między klasami i strukturami

Widziałeś, że możesz użyć klasy lub struktury do reprezentowania złożonego obiektu. Więc czego powinieneś użyć? Zaleca się używanie struktur, chyba że potrzebujesz czegoś, co wymaga klas, takich jak podklasy. W rzeczywistości pomaga to zapobiec pewnym subtelnym błędom, które mogą wystąpić, ponieważ klasy są typami referencyjnymi. Fantastyczny! Teraz, gdy znasz już klasy i struktury, przyjrzyjmy się wyliczeniom, które umożliwiają grupowanie powiązanych wartości w następnej sekcji.

Zrozumienie wyliczeń

Wyliczenia umożliwiają grupowanie powiązanych wartości, na przykład:

- kierunki kompasu (E, W, N i S)
- Kolory sygnalizacji świetlnej
- Kolory tęczy

Aby zrozumieć, dlaczego wyliczenia byłyby idealne do tego celu, rozważmy następujący przykład. Wyobraź sobie, że programujesz sygnalizację świetlną. Możesz użyć zmiennej całkowitej do reprezentowania różnych kolorów sygnalizacji świetlnej, gdzie 0 to czerwony, 1 to żółty, a 2 to zielony, w ten sposób:

```
var trafficLightColor = 2
```

Chociaż jest to możliwy sposób reprezentowania sygnalizacji świetlnej, co się stanie, gdy przypiszesz 3 do trafficLightColor? Spowoduje to problemy, ponieważ 3 nie reprezentuje prawidłowego koloru sygnalizacji świetlnej. Byłoby więc lepiej, gdybyś mógł ograniczyć możliwe wartości trafficLightColor do kolorów, które może wyświetlić. Oto jak wygląda deklaracja wyliczenia:

```
enum EnumName {

case value1

case value2

case value3

}
```

Każde wyliczenie ma opisową nazwę, a treść zawiera wartości skojarzone z tym wyliczeniem.

Zobaczmy, jak pracować z wyliczeniami. Dowiesz się, jak je tworzyć i manipulować. Zaczniesz od utworzenia takiego, który będzie reprezentował kolor sygnalizacji świetlnej w następnej sekcji.

Tworzenie wyliczenia

Stwórzmy wyliczenie reprezentujące sygnalizację świetlną. Wykonaj następujące kroki:

1. Dodaj następujący kod do swojego placu zabaw i uruchom go:

```
enum TrafficLightColor {

case red

case yellow

case green

}

var trafficLightColor = TrafficLightColor.red
```

Tworzy to wyliczenie o nazwie TrafficLightColor, które grupuje wartości czerwony, żółty i zielony. Jak widać, wartość zmiennej trafficLightColor jest ograniczona do czerwonego, żółtego i zielonego; ustawienie innej wartości spowoduje błąd.

2. Podobnie jak klasy i struktury, wyliczenia mogą zawierać metody. Dodajmy metodę do TrafficLightColor. Zmodyfikuj swój kod, jak pokazano, aby TrafficLightColor zwrócił ciąg znaków reprezentujący kolor sygnalizacji świetlnej i uruchom go:

```
enum TrafficLightColor {

case red

case yellow

case green

func description() -> String {

switch self {

case .red:

return "red"

case .yellow:

return "yellow"

default:

return "green"

}

}

}
```

```
var trafficLightColor = TrafficLightColor.red

print(trafficLightColor.description())
```

Metoda description() zwraca ciąg znaków w zależności od wartości trafficLightColor. Ponieważ wartość trafficLightColor to TrafficLightColor.red, w obszarze Debug pojawi się kolor czerwony. Wiesz już, jak tworzyć i używać wyliczeń do przechowywania zgrupowanych wartości oraz jak dodawać do nich metody. To kończy ten rozdział. Dobra robota!

Podsumowanie

Dowiedziałeś się, jak deklarować złożone obiekty przy użyciu klasy, tworzyć instancje klasy, tworzyć podklasy i przesłaniać metodę klasy. Nauczyłeś się również, jak deklarować strukturę, tworzyć instancje struktury i rozumieć różnicę między typami referencyjnymi i wartościowymi. Wreszcie nauczyłeś się, jak używać wyliczeń do reprezentowania określonego zestawu wartości. Wiesz już, jak używać klas i struktur do reprezentowania złożonych obiektów oraz jak używać wyliczeń do grupowania powiązanych wartości we własnych programach. W następnej części dowiesz się, jak określać wspólne cechy klas i struktur za pomocą protokołów, rozszerzać możliwości klas wbudowanych za pomocą rozszerzeń i jak obsługiwać błędy w programach.