GENERAL ASSEMBLY

CLASSES AND STRUCTS

Zeke Abuhoff

Lead iOS Instructor, General Assembly

CLASSES AND STRUCTS

LEARNING OBJECTIVES

+ Define classes and structs
+ Create classes and structs
+ Add properties to classes and structs
+ Add methods to classes and structs
+ Instantiate classes and structs
+ Call methods and properties

OBJECTS

Discuss:

1) Where is the code that defines a string?
2) You’re writing an app that’s a marketplace for people to buy and sell TVs. How does your code describe these TVs?
3) What are the strengths about how you describe a TV in code right now? What are the weaknesses?

OBJECTS

We can use of these types:

Bool String Int Double

Closure Array Dictionary

We know there are other types, ones that we’ve only glanced at:

UIImageView NSArray

But where do these types come from?

CLASSES AND STRUCTS

What if I told you…

CLASSES AND STRUCTS

What if I told you…

YOU can create Types?

STRUCTS

```
struct structName { }
```

Structs are objects with specific properties and methods.

Vocab

Property - a variable or constant associated with an object

Method - a function associated with an object

STRUCTS

```
struct structName { }
```

```
struct Television {

     let diagonal: Double
     let make: String
     let model: String

     func shortDescription() -> String {
          return “\(make) \(model) : \(diagonal) inches”
     }
}
```

Properties

Method

STRUCTS

Structs automatically create their own initialization method.

You can call this method to instantiate an instance of your struct.

```
// Television struct defined elsewhere

let myTelevision = Television(diagonal: 50, make: "Sony", model: "X850")
```

Vocab

Instance - one particular object of a type

Instantiate - create an instance

STRUCTS

Practice

1) Define a struct called Dog, including properties for the dog’s breed, weight and age.
2) Add a method to your Dog struct called bark, that prints a barking sound. If the dog’s weight is above a certain amount, the bark should be more of a groan.
3) Instantiate three Dog objects, each with different properties.

CLASSES AND STRUCTS

PROPERTIES AND METHODS

```
objectName.propertyName
```

```
let bigScreen = Television(diagonal: 50, make: "Vizio", model: "M")
let screenSize = bigScreen.diagonal
```

```
objectName.functionName(parameters)
```

```
let bigScreen = Television(diagonal: 50, make: "Vizio", model: "M")
let bigScreenDescription = bigScreen.shortDescription()
```

CLASSES

```
class className { }
```

Classes resemble structs in many ways. They, too, feature properties and methods.

```
class Restaurant {
      var currentPatrons = 0
      func printPatrons() {
            print(“There are \(numberOfPatrons) patrons in the restaurant.”)
      }
}
```

CLASSES

The first difference you’ll notice between classes and structs is that classes need more help with initialization.

```
class House {
    let address: String
    let numberOfRooms: Int

    init(address specifiedAddress: String, numberOfRooms specifiedNumberOfRooms: Int) {
        address = specifiedAddress
        numberOfRooms = specifiedNumberOfRooms
    }
}
```

CLASSES VS STRUCTS

What makes classes worthwhile?

If structs are better at writing their own initializers, why not always use them?

CLASSES VS STRUCTS

What makes classes worthwhile?

If structs are better at writing their own initializers, why not always use them?

The Short Answer:

Structs are passed by value.

Classes are passed by reference.

CLASSES VS STRUCTS

Passing by Value: **STRUCTS**

The computer reads the object’s value and copies it over to be the value for a new object.

Passing by Reference: **CLASSES**

The computer points the new object at the old object. Any time the new object is referenced, it returns the current value of the old object.

CLASSES VS STRUCTS

Practice:

1) Create a class called Dojo. This class should feature a variable for its address and a variable for its students.
2) Create a struct called Kata. This struct should feature a variable for a list of move names and a method that prints each move in order, following them with exclamation points.
3) Instantiate a Dojo object. Then, declare another Dojo and set it equal to the first Dojo. Change the second Dojo’s address. Print the first Dojo’s address.
4) Instantiate a Kata object. Then, declare another Kata and set it equal to the first Kata. Change the second Kata’s moves. Print the first Kata’s moves.