Mobile Application Programming: iOS

Value Types and Swift
Kevin Wong

- iOS Developer at Pixio
- Experienced with Objective-C and Swift
- Working with iOS and Android for 5~ years
- website: kwongius.github.io
Next Lecture Topics

- 3rd Party Library Management (CocoaPods, Carthage)
- More Swift Features (protocols, extensions)
- Objective-C
- Runtime (KVO, Associated Objects)
- Debugging and Testing Tools
- 3rd Party Services (Firebase)
- iOS Features (AutoLayout, MapKit)
class Character {
    var name: String
    var level: Int = 1

    func levelUp() {
        level += 1
    }

    ...

    ...
}
var characters = [Character]()

var alice = Character(name: "Alice")
characters.append(alice)

var temp = alice
temp.name = "Bob"
characters.append(temp)

alice.levelUp()
Reference Semantics

• `var temp = alice` copies the `reference`
• Both `temp` and `alice` point to the same object

```javascript
temp.name = "Bob"
```

```javascript
alice.levelUp()
```
Value Types

What is a value type?

• Used to represent a distinct value
• Equatable
• Doesn't matter how it is arrived at
• Immutable
Value Types
Base Understanding

```javascript
var value = 10
var tempValue = value
tempValue += 20

// value: 10
// tempValue: 30

tempValue == 2 * 15
```
Value Semantics

Use structs!

```swift
struct Character {
    var name: String
    var level: Int = 1

    mutating func levelUp() {
        level += 1
    }
}
```
Value Semantics

```swift
var characters = [Character]()

var alice = Character(name: "Alice")
characters.append(alice)

var temp = alice
temp.name = "Bob"

characters.append(temp)

alice.levelUp()
```
Value Semantics

var characters = [Character]()

var alice = Character(name: "Alice")

var temp = alice
temp.name = "Bob"

alice.levelUp()

characters.append(alice)
characters.append(temp)
Value Semantics

```swift
struct Character {
    var name: String
    var children: [Character] = []

    var familySize: Int {
        return 1 + children.map({ $0.familySize }).reduce(0, combine: +)
    }
}
```
Value Semantics

struct Character {
    var name: String
    var children: [Character] = []

    var familySize: Int {
        var size = 1
        for child in children {
            size += child.familySize
        }
        return size
    }
}
Value Semantics

```swift
var alice = Character(name: "Alice")
var andy = Character(name: "Andy")

alice.children.append(andy)

alice.familySize // What is family size?
```
Value Semantics

```
var alice = Character(name: "Alice")
var andy = Character(name: "Andy")

alice.children.append(andy)

alice.familySize // What is family size?
alice.children.append(alice)

alice.familySize // What is family size now?
```
Value Semantics

- Numbers are value types
- Strings have value semantics
- Arrays, Dictionaries, Sets have value semantics
Structs with References

- **NSMutableString** is a Foundation class

```swift
struct Builder {
    var name: NSMutableString
    var width: Double
    var height: Double
}
```
var builder1 = Builder()  
builder1.name = NSMutableString(string: "Builder 1")

var builder2 = builder1  
builder2.name.append(" (ver 2.0)")

var builder3 = builder1  
builder3.name = NSMutableString(string: "Builder 3")

// builder1.name == "Builder 1 (ver 2.0)"  
// builder2.name == "Builder 1 (ver 2.0)"  
// builder3.name == "Builder 3"
Make Your Structs Equatable

- Values are equatable, so your structs should be equatable
- Implement `Equatable` protocol
When to use Structs or Classes?

**Structs**

- Distinct Values
- When copies make sense
- Storage of immutable objects

**Classes**

- Inheritance
- Operations are performed
Tuples

- Tuples are ordered list of fixed elements
- Tuples have value semantics
- Tuples are equatable
Enums

- A Set of named values
- Value Type

```java
enum Suit {
    case Diamonds
    case Clubs
    case Hearts
    case Spades
}
```
Enums

define Genre {
    case Action
    case Adventure
    case Comedy
    case Documentary
    case Drama
    case Historical
    case Horror
    case SciFi
    case Western
    case Other
}
Enums

- Swift enums are extensible

```swift
enum Genre {
    var genreDescription: String {
        switch self {
        case .Action:
            return "Get your adrenaline pumping!"
        case .Comedy:
            return "Ridiculous situations to make you laugh!"
        ...
        }
    }
}
```
Enums

• Swift enums are extensible

```swift
enum Suit {
    var image: UIImage? {
        switch self {
        case .Diamonds:
            return UIImage(named: "diamonds")
        case .Clubs:
            return UIImage(named: "clubs")
        ...
        }
    }
}
```
Swift enums can be backed by values

```swift
enum ErrorCode: Int {
    case Ok = 200
    case BadRequest = 400
    case NotAuthorized = 401
    case NotFound = 404
    case Unknown = 500
}
```
Enums

- Swift enums can be backed by values
- Swift enums may have implicit values

```swift
class GameStatus: String {
    case FINISHED = "DONE"
    case WAITING
    case PLAYING
}

print(GameStatus.WAITING.rawValue)
let status = GameStatus(rawValue: "DONE")
```
 Enums

• Swift Enums allow for associated value

```swift
enum ThermostatState {
    case Cooling(Int)
    case Heating(Int)
    case Idle
    case PoweredOff
}
```
 Enums

Accessing Associated Values

switch state {
    case let .Cooling(temp):
        ...
}

if case let .Cooling(temp) = state {
    ...
}
Example