Introduction to Swift & SwiftUI

Girish Lukka

Topics to be covered

- Classes
- Structs
- Enums
- Enums
- Extensions
- Functions

Classes

- In Swift we can use classes to build custom and complex data types.
- Classes are similar to structs but are quite different in behaviour.
- No automatic memberwise initializer for your classes - need to write one.
- There is inheritance.
- A class is a blueprint for that object
- A copy of the object points to the same data by default – change one, and the copy changes too.
- An object is simply a collection of data (variables) and methods (functions).

Creating Classes

```
1
2   import Foundation
3
4   class Person{
        var clothes: String
6        var shoes: String
7   }
8
```

```
expression failed to parse:
error: ClassPH_1.xcplaygroundpage:4:7: error: class 'Person' has no initializers
class Person{
      ^
```

The properties clothes and shoes must have values.

Solution:

1. Make the values optional – messy propagation of optionals
2. Give them default values – may never be used
3. Write an initializer init() that manges the 2 properties – java class constructor

Classes Examples– Additional material P.Hudson

```swift
import Foundation
class Enemy{
    var health = 100
    var attackStrength = 10

    func move(){
        print("Walk forwards")
    }

    func attack(){

        print("Made a hit, \(attackStrength) damage")
    }
}

let enemy1 = Enemy()
print(enemy1.health)
enemy1.move()
enemy1.attack()
let enemy2  = Enemy()
```

```
100
Walk forwards
Made a hit, 10 damage
```

```swift
//inheritance

class BigEnemy : Enemy {
    var wingSpan = 2

    func talk(speech: String){
        print("Said: \(speech)")
    }
    override func move(){
        print("run fast")
    }

    override func attack(){
        super.attack()
        print("jumps, damages by 10")
    }
}
let bigEnemy = BigEnemy()
bigEnemy.wingSpan = 5
bigEnemy.attackStrength = 15
print(bigEnemy.health)
print(bigEnemy.attackStrength)
bigEnemy.talk(speech:"Hello")
bigEnemy.move()
bigEnemy.attack()
```

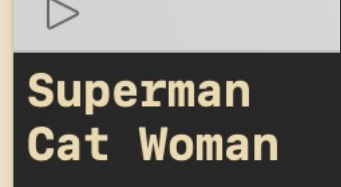# Classes Example demonstrate passing by reference

```swift
21    class Enemy1{
23        var health: Int
24        var attackStrength : Int
25
26        init(health: Int, attackStrength: Int){
27            self.health = health
28            self.attackStrength = attackStrength
29        }
30
31        func move(){
32            print("Walk forwards")
33        }
34
35        func attack(){
36
37            print("Made a hit, \(attackStrength) damage")
38        }
39
40        func takeDamage(amount: Int){
41            health = health - amount
42        }
43    }
44    let enemy1 = Enemy1(health: 10, attackStrength: 20)
45    print(enemy1.health)
46    enemy1.takeDamage(amount: 5)
47    print(enemy1.health)
48    let enemy2 = enemy1
49    print(enemy2.health)
50    // passing by reference updates for all objects.
51
```

10
5
5

Classes Example – effect of let?

```swift
class ClassHero{
    var name: String
    var location: String

    init(name:String, location: String){
        self.name = name
        self.location = location
    }
}
let classHero = ClassHero(name: "Superman", location: "Mars")
print(classHero.name)
classHero.name = "Cat Woman"
print(classHero.name)
```

```
Superman
Cat Woman
```

Structures aka Structs

Can use structs to build custom and complex data types.

Automatic generation of initializer, however, quite useful to define one.

- A struct is a blueprint for object creation
- A copy of the object is a new object – change anyone, the others unaffected, unlike class objects.
- Value type, copies the object whenever it's passed around.
- No inheritance, only encapsulation.
- Faster than classes as it doesn't require reference counting for memory management.

Struct Examples

```swift
1   import Foundation
2
3   struct Town{
4       let name: String
5       var people: [String]
6       var items: [String:Int]
7
8       func newRoad(){
9           print("New road to be built")
10      }
11  }
12
13  var myTown = Town(name: String, people: [String], items: [String : Int])
14
```

Struct Examples

```swift
1   import Foundation
2
3   struct Town{
4       let name: String
5       var people: [String]
6       var items: [String:Int]
7
8       func newRoad(){
9           print("New road to be built")
10      }
11  }
12
13  var myTown = Town(name: "MyPlace", people: ["Girish","girish"], items: ["laptops":
        2, "consoles": 3])
14
15  print(myTown) // shows all the properties and the values
16  print("\(myTown.name) has \(myTown.items["laptops"]!): laptops")
17
18  myTown.people.append("GIRISH")
19  print(myTown.people.count)
20  myTown.newRoad()
```

```
Town(name: "MyPlace", people: ["Girish", "girish"], items: ["consoles": 3, "laptops": 2])
MyPlace has 2: laptops
3
New road to be built
```

Struct examples

```swift
1   import Foundation
2
3   struct Town{
4       let name = "NewLands"
5       var people = ["Alex", "Ali"]
6       var items = ["Houses": 5, "Cars": 2, "Shops" : 1]
7
8       func newRoad(){
9           print("New road to be built")
10      }
11  }
12
13  var myTown = Town()
14
15  print(myTown) // shows all the properties and the values
16  print("\(myTown.name) has \(myTown.items["Houses"]!) houses")
17
18  myTown.people.append("Amar")
19  print(myTown.people.count)
20  myTown.newRoad()
```

```
Town(name: "NewLands", people: ["Alex", "Ali"], items: ["Cars": 2, "Houses": 5, "Shops": 1])
NewLands has 5 houses
3
New road to be built
```

Struct examples

```swift
1   import Foundation
2
3   struct Town{
4       let name : String
5       var people : [String]
6       var items : [String: Int]
7
8       init(newName: String, newPeople: [String], newItems: [String : Int])
9
10      {
11          name = newName
12          people = newPeople
13          items = newItems
14      }
15
16      func newRoad(){
17          print("New road to be built")
18      }
19  }
20  var newTown = Town(newName: "NewLands", newPeople: ["Tom", "Mark"], newItems:
        ["Homes" : 5, "Shops" : 2])
21  print(newTown)
22  newTown.people.append("Alex")
23  print(newTown.people)
```

```
Town(name: "NewLands", people: ["Tom", "Mark"], items: ["Shops": 2, "Homes": 5])
["Tom", "Mark", "Alex"]
Town(name: "NewLands", people: ["Tom", "Mark", "Alex"], items: ["Shops": 2, "Homes": 5])
Town(name: "NewLands", people: [], items: ["Shops": 2, "Homes": 5])
["Tom", "Mark", "Alex"]
```

Struct examples

```swift
1    import Foundation
2
3    struct Town{
4        let  name : String
5        var people : [String]
6        var items : [String: Int]
7
8        init(name: String, people: [String], items: [String : Int])
9        {
10           self.name = name
11           self.people = people
12           self.items  = items
13       }
14       func newRoad(){
15           print("New road to be built")
16       }
17   }
18
19   var newTown = Town(name: "NewLands", people: ["Tom", "Mark"], items: ["Homes" : 5,
20           "Shops" : 2])
20   print(newTown)
21   newTown.people.append("Alex")
22   print(newTown.people)
23   |
24   var oldTown = newTown
25   print(oldTown)
26   oldTown.people = []
27   print(oldTown)
28   print(newTown.people)
```

```
Town(name: "NewLands", people: ["Tom", "Mark"], items: ["Shops": 2, "Homes": 5])
["Tom", "Mark", "Alex"]
Town(name: "NewLands", people: ["Tom", "Mark", "Alex"], items: ["Shops": 2, "Homes": 5])
Town(name: "NewLands", people: [], items: ["Shops": 2, "Homes": 5])
["Tom", "Mark", "Alex"]
```

Struct examples mutating method

```swift
2    import Foundation
3
4    struct Town{
5        let name : String
6        var people : [String]
7        var items : [String: Int]
8
9        init(name: String, people: [String], items: [String : Int])
10       {
11           self.name = name
12           self.people = people
13           self.items  = items
14       }
15       func newRoad(){
16           print("New road to be built")
17       }
18
19        mutating func addItems(){
20           items["Hotel"] = 1
21       }
22   }
23
24   var newTown = Town(name: "NewLands", people: ["Tom", "Mark"], items: ["Homes" : 5,
         "Shops" : 2])
25   print(newTown)
26   newTown.people.append("Alex") // change from outside the struct
27   print(newTown.people)
28   newTown.addItems()
29   print(newTown)
```

```
Town(name: "NewLands", people: ["Tom", "Mark"], items: ["Homes": 5, "Shops": 2])
["Tom", "Mark", "Alex"]
Town(name: "NewLands", people: ["Tom", "Mark", "Alex"], items: ["Hotel": 1, "Homes": 5, "Shops": 2])
```

Enums example

```swift
import Foundation

struct Card {

    let rank: Rank
    let suit: Suit

    enum Rank {
        case two, three, four, five, six, seven, eigth, nine, ten, jack,
            queen, king, ace
    }

    enum Suit {
        case heart, diamond, club, spades
    }
}

struct PokerHand {
    let cards: [Card]
}
var myCard = Card(rank: .five, suit: .heart)
var myHand = PokerHand(cards: [Card(rank: .six, suit: .heart), Card(rank:
    .seven, suit: .club)])

print(myCard.suit)
print(myHand.cards[0].rank)
print(myHand.cards[0].suit)
```

Card
PokerHand

"heart\n"
"six\n"
"heart\n"

heart
six
heart
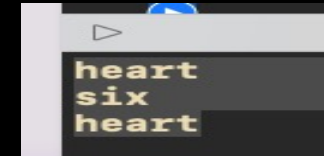
How to choose between a class, struct and enum – Apple guidelines

1. Use a struct for value types: If you want to represent a simple data structure with a few properties, a struct is the way to go. Structs are value types, which means they are copied when they are passed around in your code. This makes them suitable for small data structures that don't need to be shared between multiple instances.

2. Use a class for reference types: If you want to represent an object that can be shared between multiple instances or passed around as a reference, use a class. Classes are reference types, which means that multiple instances can reference the same object in memory.

3. Use enums for simple cases: If you want to represent a fixed set of related values, use an enum. Enums are especially useful for representing a small, finite set of values, such as the different states of a button (e.g. pressed, released).

Functions

- There are five parts to a function:

1. The keyword func. This keyword signifies the start of a function definition.

2. The name of the function.

3. The parameters (named or un-named)

4. The body of the function.

5. The return type followed by a -> eg: -> String

Function Examples

```swift
2
3    import Foundation
4
5    //  define a function like this
6    func greet(name: String) {
7        // Non-returning function
8        print("Hi \(name)!")
9    }
10
11   //  call the function like this
12   greet(name: "Girish") // prints Hi Neel!
13
14   func getFullName(firstName: String, lastName: String) ->  String {
15       // Returning function
16       return "\(firstName) \(lastName)"
17   }
18   //call function
19   let fullName = getFullName(firstName: "Girish", lastName: "Lukka")
20   print(fullName)
```

```
Hi Girish!
Girish Lukka
```

Function Examples

```swift
2
3    import Foundation
4
5    //  define a function like this
6    func greet(name: String) {
7        // Non-returning function
8        print("Hi \(name)!")
9    }
10
11   //  call the function like this
12   greet(name: "Girish") // prints Hi Neel!
13
14   func getFullName(firstName: String, lastName: String) ->  String {
15       // Returning function
16       return "\(firstName) \(lastName)"
17   }
18   //call function
19   let fullName = getFullName(firstName: "Girish", lastName: "Lukka")
20   print(fullName)
```

```
Hi Girish!
Girish Lukka
```

Function Examples

```swift
2
3    import Foundation
4
5    //  define a function like this
6    func greet(name: String) {
7        // Non-returning function
8        print("Hi \(name)!")
9    }
10
11   //  call the function like this
12   greet(name: "Girish") // prints Hi Neel!
13
14   func getFullName(firstName: String, lastName: String) ->  String {
15       // Returning function
16       return "\(firstName) \(lastName)"
17   }
18   //call function
19   let fullName = getFullName(firstName: "Girish", lastName: "Lukka")
20   print(fullName)
```

```
Hi Girish!
Girish Lukka
```

Function Examples

```swift
2
3    import Foundation
4
5    //  define a function like this
6    func greet(name: String) {
7        // Non-returning function
8        print("Hi \(name)!")
9    }
10
11   //  call the function like this
12   greet(name: "Girish") // prints Hi Neel!
13
14   func getFullName(firstName: String, lastName: String) ->  String {
15       // Returning function
16       return "\(firstName) \(lastName)"
17   }
18   //call function
19   let fullName = getFullName(firstName: "Girish", lastName: "Lukka")
20   print(fullName)
```

```
Hi Girish!
Girish Lukka
```

Function Examples

```swift
2
3  import Foundation
4
5  //  define a function like this
6  func greet(name: String) {
7      // Non-returning function
8      print("Hi \(name)!")
9  }
10
11 //  call the function like this
12 greet(name: "Girish") // prints Hi Neel!
13
14 func getFullName(firstName: String, lastName: String) ->  String {
15     // Returning function
16     return "\(firstName) \(lastName)"
17 }
18 //call function
19 let fullName = getFullName(firstName: "Girish", lastName: "Lukka")
20 print(fullName)
```

```
Hi Girish!
Girish Lukka
```

Function Examples inout

```swift
3   import Foundation
4   func doubleInPlace(number: inout Int) {
5       number *= 2
6   }
7   var myNum = 10
8   print(myNum)
9   doubleInPlace(number: &myNum)
10  print(myNum)
```

10
20

Nested Functions

```swift
func calculateMonthlyPayments(carPrice: Double, downPayment: Double,
                              interestRate: Double, paymentTerm: Double) -> Double {
    func loanAmount() -> Double {
        return carPrice - downPayment                                          ☐ (2 times)
    }
    func totalInterest() -> Double {
        return interestRate * paymentTerm                                      ☐ 24.5
    }
    func numberOfMonths() -> Double {
        return paymentTerm * 12                                                ☐ 84
    }
    return ((loanAmount() + ( loanAmount() *                                   ☐ 666.9642857..
                            totalInterest() / 100 )) / numberOfMonths())
}


let monthlyPayment = calculateMonthlyPayments(carPrice: 50000, downPayment: 5000,   ☐ 666.96428571
    interestRate: 3.5,paymentTerm: 7.0)                                                  42857

print(monthlyPayment)                                                         ☐ "666.964285..
```

Mutating functions - BMI

```swift
import Foundation

struct BMIData {
    var height = 0.0
    var weight  = 0.0
    var BMI = 0.0

    mutating func calBMI(){

        let bmiValue = weight / (height * height)
        self = BMIData(height: height, weight: weight, BMI:bmiValue)
        print("self... \(self)")

    }
}
var bmi = BMIData(height:1.67, weight: 67, BMI: 0.0)
print(bmi)
bmi.calBMI()
print(bmi)
```

```
BMIData(height: 1.67, weight: 67.0, BMI: 0.0)
self... BMIData(height: 1.67, weight: 67.0, BMI: 24.023808670084982)
BMIData(height: 1.67, weight: 67.0, BMI: 24.023808670084982)
```

Mutating functions - BMI

```swift
22  struct BMI2{
23      var height : Double
24      var weight : Double
25      var BMI = 0.0
26
27      init(height: Double, weight: Double){
28
29          self.height = height
30          self.weight = weight
31      }
32
33      mutating func calBMI2(){
34
35          let bmiValue = weight / (height * height)          24.02380867
36
37          self.BMI = bmiValue                                BMI2
38      }
39  }
40
41  var bmi2 = BMI2(height: 1.67, weight: 67)                  BMI2
42
43  bmi2.calBMI2()                                             BMI2
44  print(bmi2.BMI)                                            "24.0238086
```

Closures

- Closures are essentially anonymous functions or functions without a name and they're essentially a self-contained package of functionality that we can pass around and use.

- A normal function:

```
3    import Foundation
4
5    func functionName (parameter: parameterType) -> returnType {
6
7        //code to do something
8        return output
9    }
```

```
4
5
6    func getDrink (money: Int) -> Int {
7        let change  = money - 5                                    5
8        return change                                             5
9    }
10
11   var myChange = getDrink(money: 10)                            5
12   print(myChange)                                              "5\n"
13
```

Closures

```
2
3   import Foundation
4
5   func calculator (n1: Int, n2: Int, operation: (Int,Int) -> Int) -> Int {
6       return operation(n1,n2)                                                   (2 times)
7   }
8   //func calculator (n1: Int, n2: Int) -> Int {
9   //    return n1 + n2
10  //}
11
12  func add(no1: Int, no2: Int) -> Int {
13      return no1 + no2                                                           11
14      // signature of add is (Int,Int) -> Int
15  }
16  func times(no1: Int, no2: Int) -> Int {
17      return no1 * no2                                                          28
18      // signature of add is (Int,Int) -> Int
19  }
20  calculator(n1: 4, n2: 7, operation: add)                                      11
21  calculator(n1: 4, n2: 7, operation: times)                                   28
22
```

Closures

```swift
func calculator (n1: Int, n2: Int, operation: (Int,Int) -> Int) -> Int {
    return operation(n1,n2)
}
calculator(n1: 4, n2: 7, operation: { (no1: Int, no2: Int) -> Int in
    return no1 * no2
})
```

```swift
func calculator (n1: Int, n2: Int, operation: (Int,Int) -> Int) -> Int {
    return operation(n1,n2)
}
calculator(n1: 4, n2: 7, operation: { (no1, no2 ) -> Int in
    return no1 * no2
})
```

```swift
calculator(n1: 4, n2: 7, operation: { (no1, no2 )  in
    return no1 * no2
})
```

```swift
calculator(n1: 4, n2: 7, operation: { (no1, no2 )  in
    | no1 * no2
})
```

Automatic Reference Counting - [ARC](ARC)

Swift uses *Automatic Reference Counting* (ARC) to track and manage your app's memory usage.

```swift
class Person {
    let name: String
    init(name: String) { self.name = name }
    var apartment: Apartment?
    deinit { print("\(name) is being deinitialized") }
}

class Apartment {
    let unit: String
    init(unit: String) { self.unit = unit }
    var tenant: Person?
    deinit { print("Apartment \(unit) is being deinitialized") }
}
```
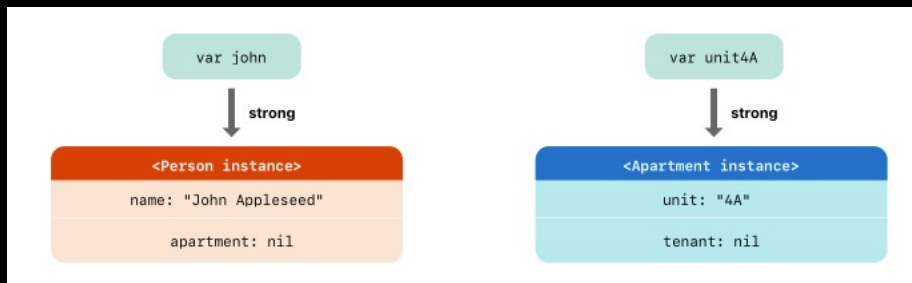

Here's how the strong references look after you link the two instances together: