iOS Dev Accelerator
Week 5 Day 5

- Week 5 in Swift
- Week Wrap up
- Watch OS with Nick

Swift Equality

- Swift's Equality has the same base concepts as Objective-C's: Equality and Identity
- What's great is, swift has an operator designed specifically for Identity checks. Its a triple equal sign ===
- Using a double equal == should make a meaningful comparison
- But there is no base class in Swift, so by default equality checks wont work on your own classes. So how can we make it work?

Equatable Protocol

- Swift has a set of protocols which define how types are compared in Swift.
- The protocols are: Equatable, Comparable, and Hashable
- Equatable : == and !=
- Comparable: <=, >, >=
- Hashable: A type must conform to hashable in order to be stored in a dictionary

Equatable Protocol

```
class Car : Equatable {
  let make : String
  let model : String

  init(make : String, model : String) {
    self.make = make
    self.model = make
  }
}

func == (lhs: Car, rhs: Car) -> Bool {
  return lhs.make == rhs.make && lhs.model == rhs.model
}

//IDENTITY
//func == (lhs: Car, rhs: Car) -> Bool {
//  return lhs === rhs
//}
```

Swift Closures and Capturing Self

- We learned that Blocks in Objective-C can create retain cycles if we are not careful
- Specifically, A block captures a strong reference to self if we access self inside the block. This will be a retain cycle if the object represented by self also keeps a strong reference to the block
- The same is true for Swift and Closures!

Weak Self in Swift

- In Objective-C, we learned we can use the __weak modifier to get a weak reference to something, and then only use weak self in the block
- In Swift, we can use something called a capture list to provide similar functionality

Capture List

```
UIView.animateWithDuration(0.3, animations: { [weak self]
  () -> Void in

})
```


```
UIView.animateWithDuration(0.3, animations: {[unowned self]
  () -> Void in

})
```

- Each item in a capture list is a paring of weak or unowned with a reference to an instance
- A capture list goes inside of []'s, before the parameters of the closure

Weak Reference

- Using the weak modifier in a capture list tells xcode that this variable may become nil at some time, so treat it as an weakly reference optional:

```
UIView.animateWithDuration(0.3, animations: { [weak self] () -> Void
   in
   if let weakSelf = self {
     weakSelf.view.alpha = 0.0
   }
})
```

Unowned Reference

- Using the unowned modifier in a capture list tells Xcode to treat this object as a weak reference rather than a strong reference, and that you are positive it will still be around by the time this closure is executed:

```
UIView.animateWithDuration(0.3, animations: {[unowned self] () -> Void
   in
   self.view.alpha = 0.0
})
```

weak vs unowned

- When to use weak vs unowned:
- Use weak when the variable could possibly become nil
- use unowned when there is no chance of that happening

Week 5 Wrap up

- Important Objective-C Concepts:
  - Declaring Classes
  - Declaring Properties
  - Calling methods with message syntax
  - Accessing properties with dot notation
  - .h vs .m
  - Blocks
  - Pointers
  - id
  - Making a class conform to a Protocol
  - Constants
  - Importing

Week 5 Wrap up

- Important iOS frameworks
  - MapKit:
    - MKMapView, MKMapViewDelegate, Annotations, Callouts
  - CoreLocation:
    - CLLocationManager, Requesting Location Services, CLLocation, CLLocationCoordinates2D

Week 5 Wrap up

- Important iOS Concepts:
- NSNotificationCenter and the Broadcast Pattern
- App States
- Retain Cycles with Blocks
- Equality
- The concept of Pointers