Announcements

• Lab 1 due on Wednesday by 11:59 PM
  – Email it to cse438ta@gmail.com

• On Wednesday we will be in Whitaker 316 for Studio 2

• Lab 2 is due on Wednesday June 28th
  – The remaining labs are posted as well

• All remaining labs are posted on the course website

Today’s Topics

• Additional Swift Concepts

• Views Introduction

• MVC

• Lab 2

• MVC and Auto Layout Demos
Lab 1 Demo

Lazy Initialization of Properties (CS193p)

- Lazy properties do not get initialized until someone accesses them
- You can allocate objects, execute a closure, or call a method

lazy var theResult = LotsOfWorkObject()

lazy var someProperty: Type = {
  // construct the value of someProperty here
  return (the constructed value)
}()

lazy var myProperty = self.initializeMyProperty()
Initialization in Swift

- Classes and structures must set all of their stored properties when created

- Various way to set properties (without an init)
  - Define default values
  - Properties may be Optional (so they start out as nil)
  - Initialize a property by setting a closure
  - Use lazy instantiation

- Use an init when values can not be set using the previous examples
  - You can have as many init methods in your class or struct
  - Each init will have different arguments

Initialization (CS193p)

- Some init methods are for free
  - Free init() given to all base classes
    - A base class has no superclass
  - If a struct has no initializers, it will get a default one will all properties as arguments
Initialization (CS193p)

- **What can I do with an init?**
  - Set property values, even those that already had defaults
  - Constant properties (those declared with let) can be set
  - You can call other init methods in your own class or struct using `self.init(args)`
  - In a class, you can also call `super.init(args)`
    - There are some rules for calling inits from other inits in a class

Class Initialization Requirements (CS193p)

- After `init` completes all properties must have values (Optionals can be nil)
- A class has two types of inits
  - Convenience and designated
- **Designated init**
  - Must (and can only) call a designated init in its immediate superclass
  - You must initialize all properties introduced by your class before calling a superclass’s `init`
  - You must call a superclass’s `init` before you assign a value to an inherited property
- **Convenience init**
  - Must (and can only) call an init in its own class
  - Must call that init before it can set any property values
  - The call of other inits must be completed before you can access properties or invoke methods
Initialization (CS193p)

• **Inheriting init**
  – If you do not implement any designated inits, you will inherit all of you superclass’s designated inits
  – If you override all of your superclass’s designated inits, you’ll inherit all its convenience inits
  – If you implement no inits, you will inherit all of your superclass’s inits
  – Any init inherited by these rules qualifies to satisfy any of the rules on the previous slide

• **Required init**
  – A class can mark one or more of its init methods as *required*
  – Any subclass must implement those init methods
    • They can be inherited per rules above

Failable init (CS193p)

• If an init is declared with a ? after the word init, it returns an Optional

init? (arg1: Type1,..) {
  // might return nil here (means init failed)
}

• **Example**  
  Let image = UIImage(named: “foo”)  //image is Optional UIImage

• **Typically use if-let for these cases**
  If let image = UIImage(named: “foo”) {  
    // image was successfully created  
  } else {  
    // failed to create image  
}
Demo

Views
View Fundamentals

- Rectangular area on screen
- Draws content
- Handles events
- Subclass of UIResponder (event handling class)
- Views arranged hierarchically
  - every view has one **superview**
  - every view has zero or more **subviews**

View Hierarchy - UIWindow

- Views live inside of a window
- UIWindow is actually just a view
  - adds some additional functionality specific to top level view
- One UIWindow for an iOS app
  - Contains the entire view hierarchy
  - Set up by default in Xcode template project
**UIView Coordinate System**

- Origin in upper left corner
- y axis grows downwards
- Units are points, not pixels
  - Points are units of coordinate system
  - Pixels are min size unit of drawing
  - Typically 2 pixels per point
    - `var ContentScaleFactor`

---

**View Controllers**
UIViewController

- Basic building block
- Manages a screenful of content
- Subclass to add your application logic

Model, View, Controller
Model, View, Controller

- **Model**
  - Manages the app data and state
  - Not concerned with UI or presentation
  - Often persists somewhere
  - Same model should be reusable, unchanged in different interfaces
View

• Present the Model to the user in an appropriate interface

• Allows user to manipulate data

• Does not store any data
  – (except to cache state)

• Easily reusable & configurable to display different data

Controller

• Intermediary between Model & View

• Updates the view when the model changes

• Updates the model when the user manipulates the view

• Typically where the app logic lives
Why Model-View-Controller?

- Separating responsibilities also leads to reusability
- By minimizing dependencies, you can take a model or view class you’ve already written and use it elsewhere
- Think of ways to write fewer lines of code
Model, View, Controller

Model Object

Controller outlets
actions

View

Lab 2 Demo
MVC Demo

Auto Layout Demo