Announcements

- Lab 2 due next Monday Sept 25th by 11:59 PM
  - Email it to cse438ta@gmail.com

- Lab 3 is posted online
  - Due October 4th

Today’s Topics

- Additional Swift Concepts
- Views
- Drawing
- Text & Images
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

```swift
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
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
  - all properties as arguments

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

View Hierarchy - Manipulation

• Add/remove views in IB or using UIView methods

  func addSubview(UIView)
  func removeFromSuperview()

• Manipulate the view hierarchy manually:

  func insertSubview(UIView, at: Int)
  func insertSubview(UIView, belowSubview: UIView)
  func insertSubview(UIView, aboveSubview: UIView)
  func exchangeSubview(at: Int, withSubviewAt: Int)
View-related Structures

- **CGPoint**
  - location in space: \( \{ x, y \} \)
  - sometimes used as an origin

- **CGSize**
  - dimensions: \( \{ \text{width}, \text{height} \} \)

- **CGRect**
  - location and dimension: \( \{ \text{origin}, \text{size} \} \)

Rects, Points and Sizes

![Diagram showing CGRect, CGPoint, and CGSize with their respective values and dimensions](image)
### View-related Structure

<table>
<thead>
<tr>
<th>Creation Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGPoint(x: Double, y: Double)</td>
<td>var point = CGPoint(x: 100.0, y: 200.0)</td>
</tr>
<tr>
<td></td>
<td>point.x = 300.0</td>
</tr>
<tr>
<td></td>
<td>point.y = 30.0</td>
</tr>
<tr>
<td>CGSize(width: Double, height: Double)</td>
<td>var size = CGSize(width: 42.0, height: 11.0);</td>
</tr>
<tr>
<td></td>
<td>size.width = 100.0</td>
</tr>
<tr>
<td></td>
<td>size.height = 72.0</td>
</tr>
<tr>
<td>CGRect(x: Double, y: Double, width: Double, height: Double)</td>
<td>var rect = CGRect(x: 100.0, y: 200.0, width: 42.0, height: 11.0)</td>
</tr>
<tr>
<td></td>
<td>rect.origin.x = 0.0</td>
</tr>
<tr>
<td></td>
<td>rect.size.width = 50.0</td>
</tr>
</tbody>
</table>

### 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
### Location and Size

- View's location and size expressed in two ways
  - Frame is in superview's coordinate system
  - Bounds is in local coordinate system

#### View A
- Frame:
  - Origin: 0,0
  - Size: 550 x 400

#### View A bounds:
- Origin: 0,0
- Size: 550 x 400

#### View B
- Frame:
  - Origin: 200, 100
  - Size: 200 x 250

#### View B bounds:
- Origin: 0,0
- Size: 200 x 250

What about View B?

### Frame and Bounds

- Which to use?
  - Usually depends on the context

- If you are using a view, typically you use bounds

- If you are implementing a view, typically you use frame

- Matter of perspective
  - From outside it's usually the frame
  - From inside it's usually the bounds

- Examples:
  - Creating a view, positioning a view in superview - use frame
  - Handling events, drawing a view - use bounds
Creating Views

Where do views come from?

- Commonly placed in Storyboard
- Drag out any of the existing view objects (buttons, labels, etc)
- Or drag generic UIView and set custom class
Manual Creation

- Views are initialized using `UIView.init(frame:)`
  ```swift
  let theFrame = CGRect(x:0, y:0, width:200, height:150)
  let myView = UIView(frame: theFrame)
  ```

- Example:
  ```swift
  let frame = CGRect(x:20, y:45, width: 140, height: 20)
  let myLabel = UILabel(frame:frame)
  myLabel.text = "Hello Class"
  view.addSubview(myLabel)
  ```

Defining Custom Views

- Subclass `UIView`

- For custom drawing, you override:
  ```swift
  func draw(_ rect: CGRect)
  ```

- For event handling, you override:
  ```swift
  func touchesBegan(_ touches: Set<UITouch> withEvent:UIEvent?)
  func touchesMoved(_ touches: Set<UITouch> withEvent:UIEvent?)
  func touchesEnded(_ touches: Set<UITouch> withEvent:(UIEvent?))
  ```
Drawing Views

**draw: Method**

- **draw: does nothing by default**
  - If not overridden, then backgroundColor is used to fill

- **Override – draw: to draw a custom view**
  - rect argument is area to draw

- **When is it OK to call draw:?**
Be Lazy

- **draw: is invoked automatically**
  - Don’t call it directly!

- **Being lazy is good for performance**

- **When a view needs to be redrawn, use:**
  setNeedsDisplay

Demo
CoreGraphics and Quartz 2D

- UIKit offers very basic drawing functionality
  - UIRectFill(CGRect rect)
  - UIRectFrame(CGRect rect)

- CoreGraphics: Drawing APIs

- CG is a C-based API, not Objective-C

- CG and Quartz 2D drawing engine define simple but powerful graphics primitives
  - Graphics context
  - Transformations
  - Paths
  - Colors
  - Fonts
  - Painting operations

CG Wrappers

- Some CG functionality wrapped by UIKit
- UIColor
  - Convenience for common colors
  - Easily set the fill and/or stroke colors when drawing
    ```
    UIColor.red.set()
    // drawing will be done in red
    ```

- UIFont
  - Access system font
  - Get font by name
  - Get preferred font for a given text style
    - Best way for font in code
      ```
      class func preferredFont(forTextStyle style: UIFontTextStyle) -> UIFont
      ```
  - A few examples of Text Styles
    - UIFontTextStyle.headline
    - UIFontTextStyle.body
    - UIFontTextStyle.footnote
**Simple draw(_:)** example

- **Draw a solid color and shape**

```swift
override func draw(_ rect: CGRect) {
    let bounds = self.bounds

    UIColor.gray.set()
    UIRectFill(bounds)

    let myShape = CGRect(x: 10, y: 10, width: 50, height: 100)
    UIColor.red.set()
    UIRectFill(myShape)

    UIColor.black.set()
    UIRectFrame(myShape)
}
```

**What shape is this?**

---

**Drawing More Complex Shapes**

- **Common steps for draw:**
  - Get current graphics context
  - Define a path
  - Set a color
  - Stroke or fill path
  - Repeat, if necessary
Paths

- CoreGraphics paths define shapes
- Made up of lines, arcs, curves and rectangles
- Creation and drawing of paths are two distinct operations
  - Define path first, then draw it

Drawing Shapes using Bezier Paths

- First create a Bezier Path
  let path = UIBezierPath()

- Move around, add lines or arcs to path
  path.move(to: CGPoint(x:60,y:40))
  path.addLine(to: CGPoint(x:100,y:50))
Simple Example

```swift
override func draw(_ rect: CGRect){
    let path = UIBezierPath()
    path.move(to: CGPoint(x: 75, y: 10))
    path.addLine(to: CGPoint(x: 10, y: 150))
    path.addLine(to: CGPoint(x: 160, y: 150))
    path.close()
    UIColor.red.setFill()
    UIColor.black.setStroke()
    path.lineWidth = 3.0
    path.stroke()
    path.fill()
}
```

What shape is this?

More Drawing Information

- UIView Class Reference
- CGContext Reference
- “Quartz 2D Programming Guide”
- Lots of samples in the iPhone Dev Center
Lab 3 Preview

Additional Examples