Building an iPhone Application

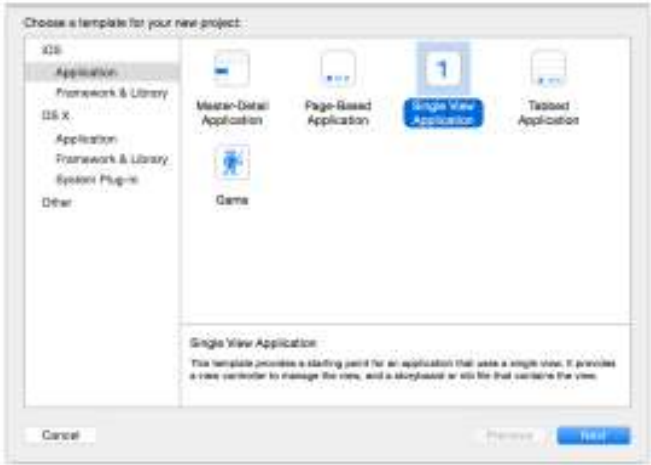
448460-1
Fall 2015
10/05/2015
Kyoung Shin Park
Multimedia Engineering
Dankook University

Overview

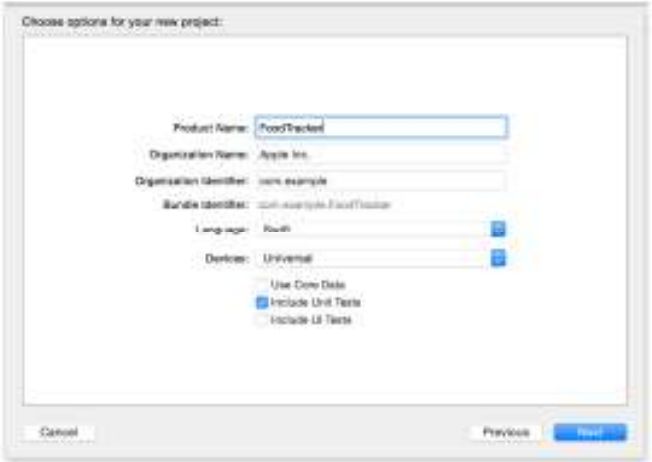
- Building an iOS Application (Swift)
- Model-View-Controller Design
- Interface Builder and Nib Files
- Controls and Target-Action

Create a New Project

Create a New Project



Create a New Project

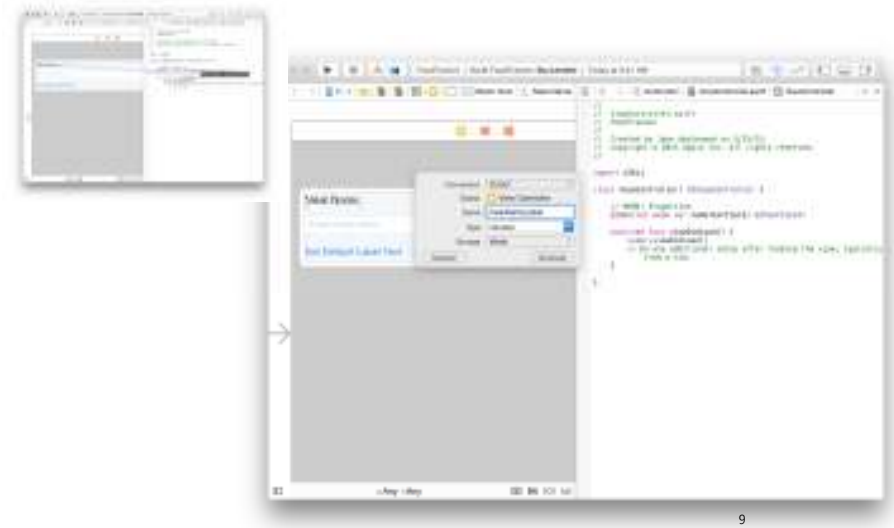


Xcode

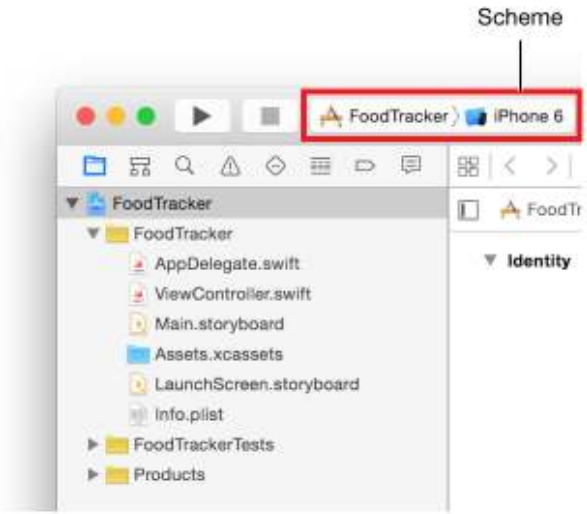
Open your Storyboard

Build the Basic UI

Connect the UI to Code



Running a Simulator



Building an iOS Application (Swift)

AppDelegate.swift

- AppDelegate.swift create **the entry point** to your app and a run loop that delivers input events to your app.
  - **@UIApplicationMain** attribute **creates an application object** that is responsible for managing the life cycle of the app and **app delegate object**.
  - **AppDelegate** class contains a single property: **window**.

    ```
    var window: UIWindow?
    ```
  - **AppDelegate** class also contains template implementations of important methods.

    ```
    func application(application: UIApplication,
    didFinishLaunchingWithOptions launchOptions: [NSObject:
    AnyObject]?) -> Bool
    func applicationWillResignActive(application: UIApplication)
    func applicationDidEnterBackground(application: UIApplication)
    func applicationWillEnterForeground(application: UIApplication)
    func applicationDidBecomeActive(application: UIApplication)
    func applicationWillTerminate(application: UIApplication)
    ```

ViewController.swift

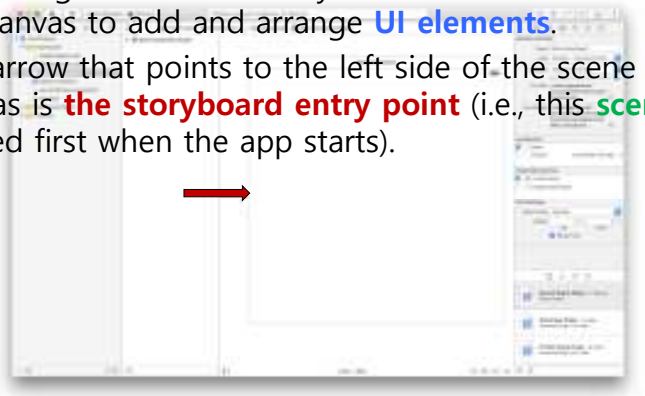
- A custom subclass of **UIViewController** named **ViewController**.
  - You **override** the methods defined on UIViewController, such as **viewDidLoad()** and **didReceiveMemoryWarning()**.

```
// After instantiation and outlet setting, viewDidLoad is called
override func viewDidLoad() {
    super.viewDidLoad()
    // do any additional setup of my app, typically from a nib
}
```

  - You **implement your own custom methods.**

Main.storyboard

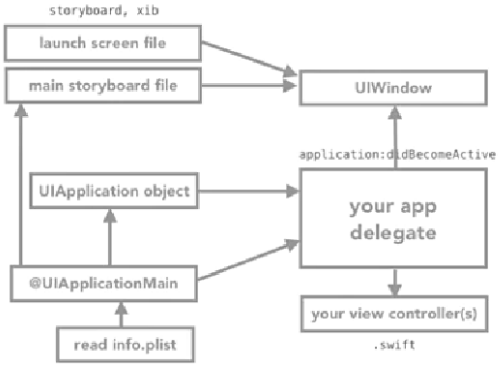
- A **storyboard** is a visual representation of the app's user interface. You use storyboards to lay out the flow (or story) that drives your app.
- The background of the storyboard is the **canvas**. You use the canvas to add and arrange **UI elements**.
- The arrow that points to the left side of the scene on the canvas is **the storyboard entry point** (i.e., this **scene** is loaded first when the app starts).



Anatomy of an iOS Application

- Compiled code
  - Your code
  - Framework
- Nib files
  - UI elements and other objects
  - Details about object relationships
- Resources (images, sounds, strings, etc)
- Info.plist file (application configuration)

iOS Application Lifecycle in Swift



UIKit Framework

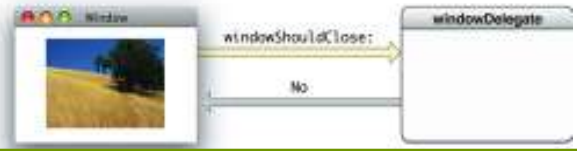
- UIKit provides standard interface elements
  - button, label, slider, tableview, etc
- Every application has a single instance of UIApplication
  - Singleton design pattern

```
let app = UIApplication.sharedApplication()
```

  - Orchestrates the lifecycle of an application
  - Dispatches events
  - Manages status bar, application icon badge
  - Rarely subclassed; **Uses delegation instead**

Delegation

- Delegate allows one object to act on behalf of another object
- Control passed to **delegate** objects to perform application specific behavior
- Avoids need to subclass complex objects
- Many UIKit classes use delegates
  - UIApplication
  - UITableView
  - UITextField



The delegate is automatically registered as an observer of notifications posted by the delegating object. The delegate need only implement a notification method declared by the framework class to receive a particular notification message. This window object posts an **NSWindowWillCloseNotification** to observers, but sends a **windowShouldClose:** message to its delegate.

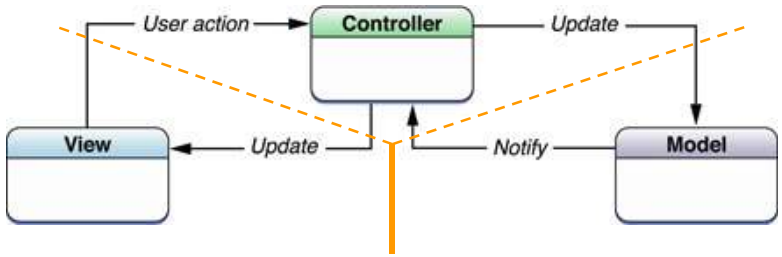
Info.plist file

- Property List (often XML), describing your application
  - Icon appearance
  - Status bar style (default, black, hidden)
  - Orientation
  - Uses Wifi networking
  - System Requirements
- Can edit most properties in Xcode by clicking on Info.plist
- Can edit it as raw XML by Opening As Source Code.
- Usually you edit Info.plist settings by clicking on your project in the Navigator.

Model View Controller

Model View Controller

- The Model-View-Controller (MVC) design pattern assigns objects in an application one of three roles: model, view, or controller.



Model = **What** you application is (but **not how** it is displayed)
Controller = **How** your Model is presented to the user (UI logic)
View = Your Controller's minions

Model

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

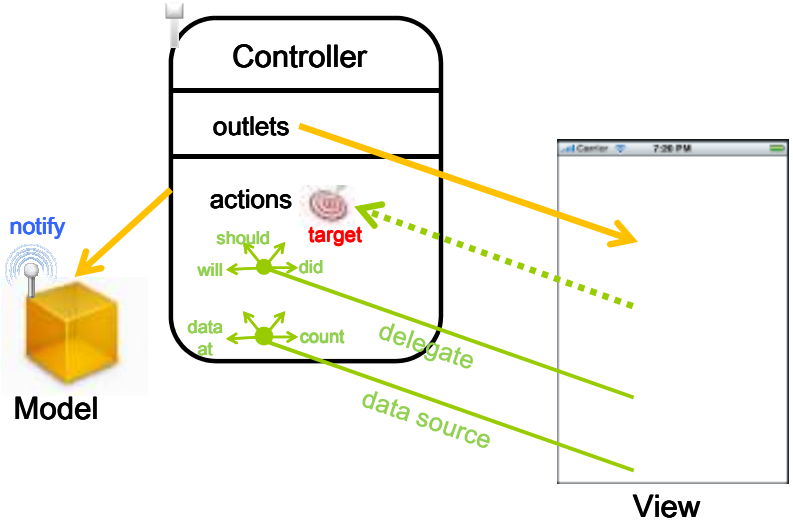
View

- 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 application logic lives

Model View Controller



Interface Builder and Nib

Nib Files

- Helps you design the View in MVC
  - Layout user interface elements
  - Add controller objects
  - Connect the controller and UI

Nib Loading

- At runtime, objects are unarchived
  - Values/settings in Interface Builder are restored
  - Ensures all outlets and actions are connected
  - Order of unarchiving is not defined
- If loading the nib automatically creates objects and order is undefined, how do I customize?
  - awakeFromNib

awakeFromNib

- awakeFromNib method is sent to all objects that come out of a storyboard (including your Controller).
- It happens before outlets are set (i.e., before the MVC is loaded).
- You should put your code somewhere else if at all possible (e.g., viewDidLoad or viewWillAppear)

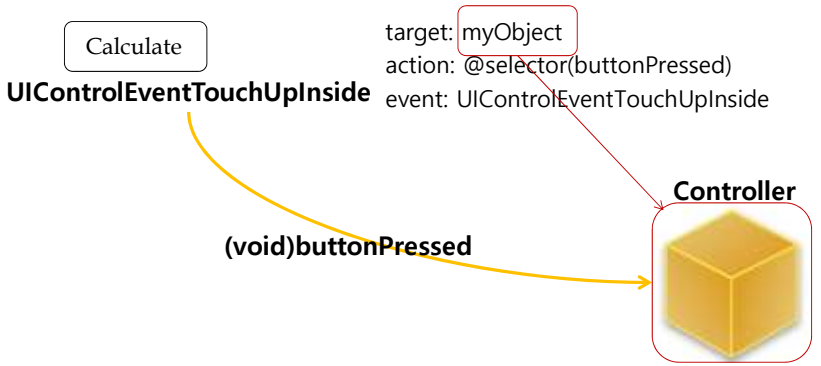
Controls and Target/Action

Controls – Events

- View objects that allow users to initiate some type of action
- Respond to variety of events
  - Touch events
    - touchDown
    - touchDragged (entered, exited, drag inside, drag outside)
    - touchUp
  - Value changed
  - Editing events
    - editing began
    - editing changed
    - editing ended

Controls – Target/Action

- When event occurs, actions is invoked on target object



Action Methods

- 3 different flavors of action method selector types
  - Simple no-argument selector

```
func increase() { // bump the number of sides of the polygon up
    polygon.numberOfSides += 1
}
```

  - Single argument selector control is 'sender'

```
func adjustNumberOfSides(sender: AnyObject) { // if it is a slider
    if let slider = sender as? UISlider {
        polygon.numberOfSides = slider.value
    }
}
```

  - Two arguments in selector (sender & event)

```
func touchesBegan(touches: Set<NSObject>, withEvent event: UIEvent)
{ ... }
```

- UIEvent contains details about the event that took place

Multiple Target-Actions

- **Contols** can **trigger** multiple **actions** on different targets in response to the same event
- Different than Cocoa on the desktop where only one target actions is supported
- Different events can be setup in Interface Builder

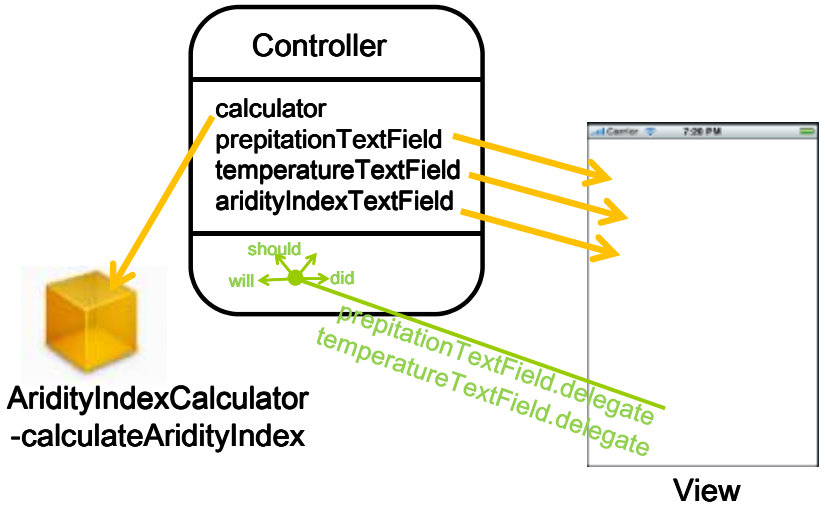
Delegation

Delegation

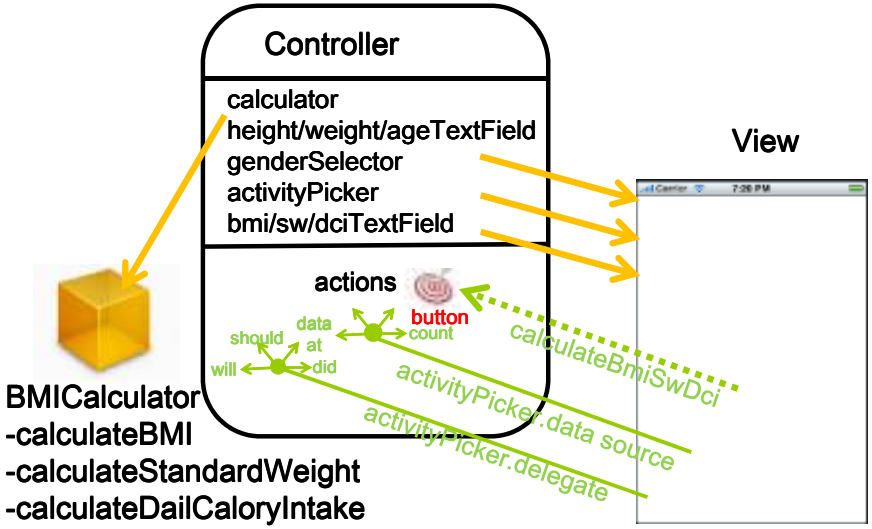
- **Control** passed to **delegate** objects to perform application specific behavior
- How it plays out
  - Create a **delegation protocol** (defines what the View wants the Controller to take care of)
  - Create a **delegate** property in the View whose type is that delegation protocol
  - Use the **delegate** property in the View to get/do things it can't own or control
  - Controller declares that it **implements the protocol**
  - Controller sets **self** as the delegate of the View by setting the delegate property
  - Implement the protocol in the Controller

Demo

Model View Controller



Model View Controller



Views

View Fundamentals

- A **view** (i.e., **UIView subclass**) represents a rectangular area on screen
- Draws content and handles events in that rectangle
- Subclass of **UIResponder** (event handling class)
- Views arranged hierarchically
  - Every view has only **one superview – var superview: UIView?**
  - Every view has **zero or more subviews – var subviews: [UIView]**
    - It's actually [AnyObject]
  - Subview order (in that array) matters: those later in the array are on top of those earlier
  - A view can clip its subviews to its own bounds or not (the default is not to)

View Hierarchy - UIWindow

- Views live inside of a window
- **UIWindow** is actually just a view
  - Adds some additional functionality specific to top level view
- **Usually only one UIWindow** for an iPhone application
  - Contains the entire view hierarchy
  - Set up by default in Xcode template project

View Hierarchy - Manipulation

- Hierarchy is most often constructed in Xcode graphically
  - Even custom views are usually added to the view hierarchy using Interface Builder
- It can be done in code using UIView methods

  addSubview(aView: UIView) // sent to aView's superview

  removeFromSuperview() // sent to the view you want to remove
- Manipulate the view hierarchy manually

  insertSubview: atIndex:

  insertSubview: belowSubview:

  insertSubview: aboveSubview:

  exchangeSubviewAtIndex: withSubviewAtIndex:

View Hierarchy

- **Where does the view hierarchy start?**
  - The top of the (useable) view hierarchy is the Controller's **var view: UIView**
  - This simple property is a very important thing to understand!
  - This view is the one whose bounds will change on rotation, for example.
  - This view is likely the one you will programmatically add subviews to (if you ever do that).
  - All of your MVC's View's UIViews will have this view as an ancestor.
  - It's automatically hooked up for you when you create an MVC in Xcode.

Initializing a UIView

- A UIView's initializer is different if it comes out of a storyboard

```
init(frame: CGRect) // initializer if the UIView is created in code
init(coder: NSCoder) // if the UIView comes out of a storyboard
```

- If you need an initializer, implement them both

```
func setup() { ... }
override init(frame: CGRect) { // designated initializer
    super.init(frame: frame)
    setup()
}
required init?(coder aDecoder: NSCoder) {// required initializer
    super.init(coder: aDecoder)
    setup()
}
```

Initializing a UIView

- Another alternative to initializers in UIView
  - **awakeFromNib() // this is only called if the UIView came out of a storyboard**
  - This is not an initializer (it's called immediately after initialization is complete).
  - All objects that inherit from NSObject in a storyboard are sent this (if they implement it).
  - Order is not guaranteed, so you cannot message any other objects in the storyboard here.

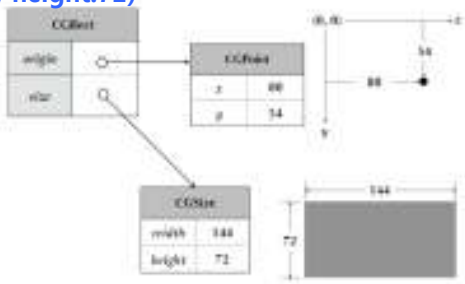
View-related Data Structures

- CGFloat
  - Always use this instead of Double or Float for anything to do with a UIView's coordinate system, **let val = CGFloat(doubleVal)**
- CGPoint

```
var point = CGPoint(x: 80, y: 54)
```

- CGSize

```
var size = CGSize(width: 144, height:72)
size.width += 42.5
size.height += 75
```



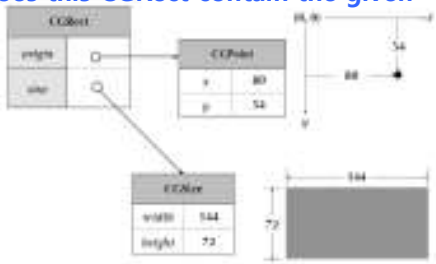
View-related Data Structures

- CGRect

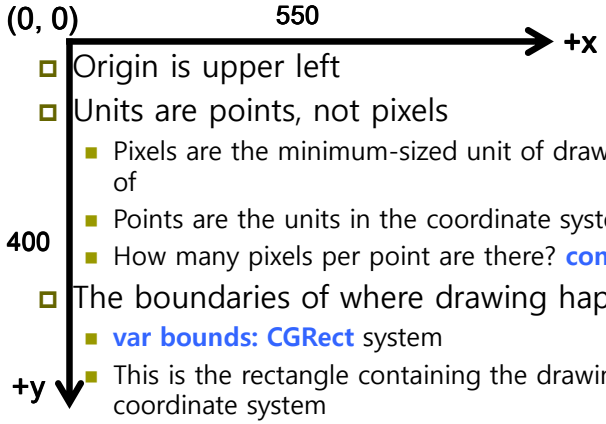
```
let rect = CGRect(origin: point, size: size)
```

- Lots of convenient properties and functions on CGRect

```
var minX: CGFloat // left edge
var midY: CGFloat // midpoint vertically
intersects(CGRect) -> Bool // does this CGRect intersect this other one?
contains(CGPoint) -> Bool // does this CGRect contain the given point?
```

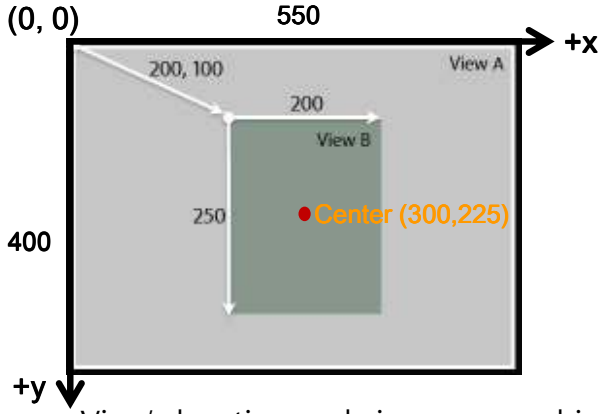


UIView Coordinate System



- Origin is upper left
- Units are points, not pixels
  - Pixels are the minimum-sized unit of drawing your device is capable of
  - Points are the units in the coordinate system
  - How many pixels per point are there? **contentScaleFactor: CGFloat**
- The boundaries of where drawing happens
  - **var bounds: CGRect** system
  - This is the rectangle containing the drawing space in its own coordinate system
- Where is the UIView?
  - **var center: CGPoint** // the center of a UIView (superview's coord)
  - **var frame: CGRect** // the rect containing a UIView (superview's coor

UIView Coordinate System

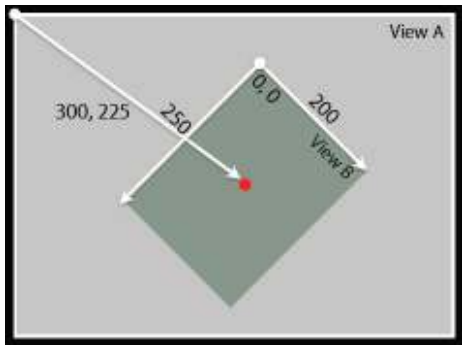


View A Frame:
Origin: (0, 0)
Size: 550 x 400
View A Bounds:
Origin: (0, 0)
Size: 550 x 400
View B Frame:
Origin: (200, 100)
Size: 200 x 250
View B Bounds:
Origin: (0, 0)
Size: 200 x 250

- View's location and size expressed in two ways:
  - **Frame** is in superview's coordinate system
  - **Bounds** is in local coordinate system
  - **Center** is the center of your view in your superview's coordinates

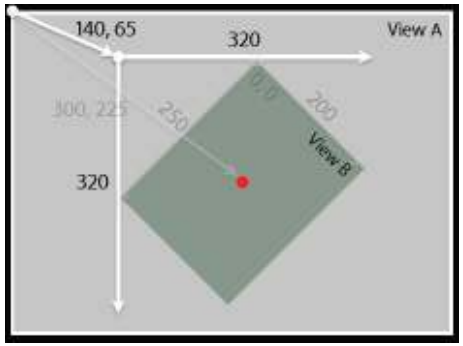
Transform

- 45° Rotation



Frame

- The smallest rectangle in the superview's coordinate system that fully encompasses the view itself



View B Center:
Origin: (300, 225)
View B Frame:
Origin: (145, 65)
Size: 320 x 320
View B Bounds:
Origin: (0, 0)
Size: 200 x 250

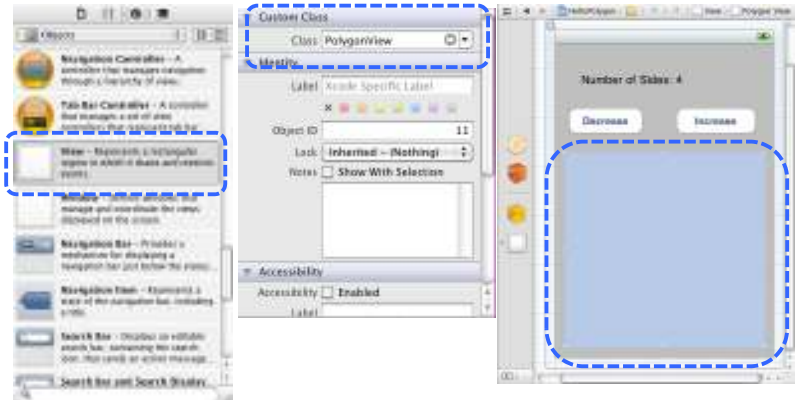
Frame and Bounds

- If you are using a view, typically you use frame
- If you are implementing a view, typically you use bounds
- 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?

- Most often your views are created via your storyboard
  - Xcode's Object Paletter has a generic UIView you can drag out
  - After you do that, you must use Identity Insepctor to changes its class to your subclass



Manual Creation

- You can create a UIView via code

  let myView = UIView(frame: myFrame) // frame initializer
- Example

  let labelRect = CGRect(x: 20, y:20, width: 100, height: 50)

  let label = UILabel(frame: labelRect) // UILabel is a subclass of UIView

  label.text = "Number of sides: "

  view.addSubview(label)

Defining Custom Views

- When to create my own **UIView subclass**?
  - I want to do some custom drawing on screen
- For custom drawing, you override
  - **override func drawRect(regionThatNeedsToBeDrawn: CGRect)**
- **Never call drawRect!!** Instead, if your view needs to be redrawn, let the system know that by calling
  - **setNeedsDisplay()**
  - **setNeedsDisplayInRect(regionThatNeedsToBeRedrawn: CGRect)**
- For example (PolygonView.m)

```
func setNumberOfPolygonSides(sides: Int) {
    numberOfSides = sides
    self.setNeedsDisplay()
}
```

Drawing Views

CoreGraphics

- UIKit offers very basic drawing functionality
  - **UIRectFill(CGRect rect);**
  - **UIRectFrame(CGRect rect);**
- CoreGraphics (CG): Drawing APIs
  - CG is a C-based (non object-oriented) API
  - CG drawing API define simple but powerful graphics primitives
    - Graphics context
    - Transformations
    - Paths
    - Colors
    - Fonts
    - Painting operations

CoreGraphics Concepts

- Common steps for drawRect: are
  - You get a graphics context to draw into (could be printing context, drawing context, etc). The function **UIGraphicsGetCurrentContext()** gives a context you can use in drawRect
  - Create **paths** (out of lines, arcs, transform, etc)
  - Set drawing attributes like colors, fonts, textures, linewidths, linecaps, etc
  - Stroke or fill the created paths with the given attributes

Paths

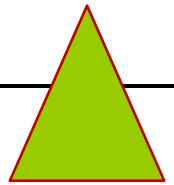
- CoreGraphics paths define shapes
- Made up of lines, arcs, curves and rectangles
- Creation and drawing of paths are two distinct operations
  - Define path first, then draw it
- Two parallel sets of functions for using paths
  - CGContext "convenience" throwaway functions
  - CGPath functions for creating reusable paths

CGContext	CGPath
CGContextMoveToPoint	CGPathMoveToPoint
CGContextAddLineToPoint	CGPathAddLineToPoint
CGContextAddArcToPoint	CGPathAddArcToPoint
CGContextClosePath	CGPathSubPath
and so on.....	

UIBezierPath

- Object-oriented **UIBezierPath** class
  - Same as core graphics, but captures all the drawing with a UIBezierPath instance
  - UIBezierPath automatically draws in the "current" context (drawRect sets this up for you)
  - Methods for adding to the UIBezierPath (lineto, arcs, etc) and setting linewidth, etc
  - Methods for stroke or fill the UIBezierPath

Simple Path Example



```
// draw a shape and path
override func drawRect(rect: CGRect) {
    let path = UIBezierPath() // create a UIBezierPath
    path.moveToPoint(CGPoint(80, 50)) // assume screen is 160x250
    path.addLineToPoint(CGPoint(140,150))
    path.addLineToPoint(CGPoint(10,150))
    path.closePath() // close the path
    UIColor.greenColor().setFill() // set attributes & stroke/fill
    UIColor.redColor().setStroke() // a method in UIColor
    path.lineWidth = 3.0 // a property in UIBezierPath
    path.fill() // fill with green color
    path.stroke() // stroke line with red color
}
```

Drawing

- You can also draw common shapes with UIBezierPath

  ```
  let roundRect = UIBezierPath(roundedSet: aCGRect,
  cornerRadius: aCGFloat)
  let oval = UIBezierPath(ovalInRect: a CGRect)
  ```

- Clipping your drawing to a UIBezierPath's path

  addClip() // you could clip to a rounded rect to enforce the edges of a playing card

- **Hit detection**

  func containsPoint(CGPoint) -> Bool // returns whether the point is inside the path (the path must be closed. The winding rule can be set with usesEvenOddFillRule property.)

UIColor

- Colors are set using UIColor
  - There are type methods for standard colors, e.g. **let greenColor = UIColor.greenColor()**

  myLabel.textColor = UIColor.blueColor() // blue label text
  - You can also create them from RGB, HSB, or even a pattern (using UIImage)
- Background color of a UIView

  var backgroundColor: UIColor
- Colors can have alpha (transparency)

  let transparentWhite = UIColor.whiteColor().colorWithAlphaComponent(0.5) // alpha is between 0.0 (fully transparent) and 1.0 (fully opaque)

  view.backgroundColor = transparentWhite // set background color of view to the UIColor with alpha

View Transparency

- What happens when views overlap and have transparency?
  - **Subviews** list order determines who is in front
  - Lower ones (earlier in the array) can "show through" transparent views on top of them
  - Transparency is not cheap, by the way, so use it wisely
- When you are drawing, you can draw with transparency
  - By default, drawing is full opaque!
- You can hide a view completely without removing it from view hierarchy

  var hidden: Bool
  - A hidden view will draw nothing on screen and get no events either
  - Not as uncommon as you might think to temporarily hide a view

UIFont

- Fonts are set using UIFont

  myLabel.font = UIFont(name: "Helvetica", size: CGFloat(20))
- To get preferred font for a given text style using UIFont type method

  class func preferredFontForTextStyle(UIFontTextStyle) -> UIFont
  - Some of the styles (see UIFontDescriptor documentation)

  UIFontTextStyle.Headline

  UIFontTextStyle.Body

  UIFontTextStyle.Footnote
- There are also "system fonts"

  class func systemFontOfSize(pointSize: CGFont) -> UIFont

  class func boldSystemFontOfSize(pointSize: CGFont) -> UIFont

Images & Text

Drawing Text

- Usually we use a UILabel to put text on screen
  - if we want to draw text in our **drawRect**

```
let color: UIColor = UIColor.darkGrayColor() // color
let font = UIFont(name: "Helvetica Neue", size: 18) // font
var paraStyle = NSMutableParagraphStyle() // line spacing
paraStyle.lineSpacing = 6.0
let skew = 0.1 // obliqueness
let baselineAdjust = 1.0
var attributes: NSDictionary =
[ NSForegroundColorAttributeName: color,
NSFontAttributeName: font, NSParagraphStyleAttributeName:
paraStyle, NSObliquenessAttributeName: skew,
NSBaselineOffsetAttributedName: baselineAdjust]
let text: NSString = "hello"
text.drawInRect(CGRectZero, withAttributes: attributes)
```

Drawing Images

- There is a UILabel-equivalent for images: **UIImageView**
  - But, you might want to draw the image inside your drawRect
- Creating a UIImage object

```
let image: UIImage? = UIImage(named: ""foo") // optional
```

  - You add foo.jpg to your project in the **Images.xcassets** file
  - Images will have different resolutions for different devices (all managed in Images.xcassets)
- You can also create one from files in the file system

```
let image: UIImage? = UIImage(contentsOfFile: aString)
let image: UIImage? = UIImage(data: anNSData) // raw jpg, png, tiff, etc
```

- You can even create one by drawing with Core Graphics
  - **UIGraphicsBeginImageContext(CGSize)**

Drawing Images

- Once you have a UIImage, you can blast its bits on screen

```
let image: UIImage = ....
image.drawAtPoint(aCGPoint) // upper left corner
image.drawInRect(aCGRect) // scales the image to fit a CGRect
image.drawAsPatternInRect(aCGRect) // tiles the image
```

References

- Lecture 5&6&8&13 Slide from Developing iOS8 Apps with Swift (Winter 2015) @Stanford University
- https://developer.apple.com/library/prerelease/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/index.html#//apple_ref/doc/uid/TP40015214-CH2-SW1
- https://developer.apple.com/library/prerelease/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/Lesson2.html#//apple_ref/doc/uid/TP40015214-CH5-SW1