GENERAL ASSEMBLY

MOBILE DEVELOPMENT

POWERING APPS WITH CODE

William Martin
Head of Product, Floored

LEARNING OBJECTIVES

- Create IB "hooks" from UI elements to Swift code.
- Identify and override methods associated with View Controllers inherited from their superclasses.
- Identify how to access documentation for IB elements and controllers.

REVIEW

WHERE HAVE WE BEEN?

VALUES AND TYPES

```
// Integers
–1, 0, 1, 2, 3
// Doubles (and Floats)
3.14159265358979
// Bools
true and false
// Strings
“Hello, Toshi!”
```

SYNTAX: VARIABLES

```
// Declare and initialize
var myName : String = “Toshi”

// Assign a variable
myName = “Toshi the Shiba”
```

SYNTAX: CONDITIONALS (IF-STATEMENTS)

```
if x > 0 {
    print(“positive”)
} else if x < 0 {
    print(“negative”)
} else {
    print(“zero”)
}
```

SYNTAX: LOOPS (WHILE)

```
var x = 1
while x < 15 {
    print(“x is \(x)”)
    x++
}
```

SYNTAX: LOOPS (FOR)

```
for (var x = 0; x < 10; x+=2) {
    print(“x = \(x)!”)
}

for x in 0..<10 {
    print(“x = \(x)!”)
}
```

SYNTAX: FUNCTIONS

```
func isPrime(number:Integer) -> Boolean {
    for i in 2..<number {
        if number % i == 0 {
            return false
        }
    }
    return true
}
```

SYNTAX: CLASSES

```
class Dog {
    var name : String
    init(name:String) {
        self.name = name
    }
    func bark() {
       print(“\(self.name) says BARK!”)
    }
}
```

SYNTAX: CLASSES

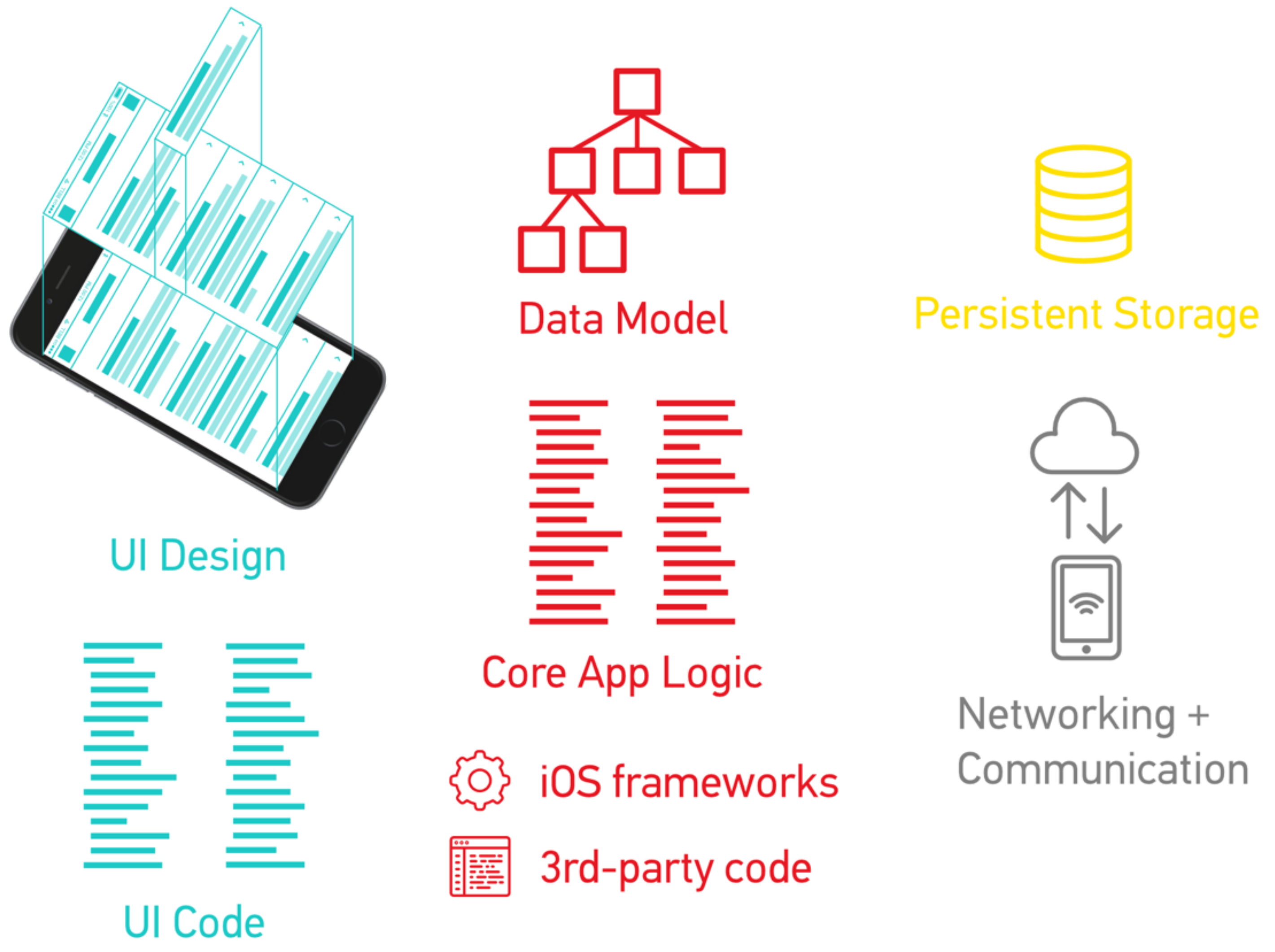
```
// Instantiate
var myDog : Dog = Dog(name:“Toshi”)

// Access properties and methods
print(myDog.name)
myDog.bark()
```

SYNTAX: EXTENDING CLASSES

```
class Shiba : Dog {
    var temperament : String = “stubborn”
    override func bark() {
        print(“Shibas don’t bark.”)
    }
}
```

INTERFACE BUILDER + CODE

WHERE DOES CODE LIVE IN AN IPHONE APP?



WHERE DOES CODE LIVE IN AN IPHONE APP?

- AppDelegate
- View Controllers (subclasses of UIViewController)
- Custom Classes (your own .swift files)
- Custom Views (subclasses of UIView and other views)

APPDELEGATE

- Controls app-level events.
- For example, what to do when:
  - an app is launched from the home screen,
  - an app goes into the "background," e.g. when receiving a phone call or pressing the home button,
  - an app becomes active,
  - etc.

VIEW CONTROLLERS

- Each represents a single scene (or screen).
- Controls the behaviors of the views on the screen.
- Has access to the application's data, so it knows what info to display.
- Handles what to do when the user takes action (e.g. taps, swipes, etc.).
- Subclasses of UIViewController, a class provided by the UIKit framework.
- All have a .view property that includes a default UIView.

CUSTOM VIEWS

- Take existing views provided by UIKit and augment their appearance and behaviors.
- e.g. a UITextView with special text highlighting.

CUSTOM CLASSES

- Data model
  - The classes necessary to describe the user data stored and manipulated by the app.
  - e.g. User, Task, Project, etc.
- Object model
  - Any additional classes needed in the implementation.
  - e.g. Network connection manager, etc.
- Helper code
  - Code that makes your life easier.

LET'S TAKE A TOUR OF VIEWCONTROLLER.SWIFT

- Create an Xcode project.
  - Template: Single View Project
  - Disable Autolayout in Main.storyboard
- ViewController.swift includes the code behind "View Controller" in the storyboard.

INTERFACE BUILDERS'S "HOOKS"

There are two "hooks" we can use to work with Views that are placed in a Storyboard with Interface Builder. They are:

- **IBActions** – methods called by user-driven events, like button taps.
- **IBOutlets** – references (instances) to elements in the UI.

XCODE: IBACTION / IBOUTLET DEMO

COLOR SLIDERS APP

POWERING APPS WITH CODE

EXERCISE

KEY OBJECTIVE(S)

Make an app that adjusts a view's parameters with user interface elements and Swift code.

TIMING

30 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

Create an app with a single screen and the following views:

- Single UIView rectangle with a colored background.
- Three sliders, one each that adjusts the hue, brightness, and saturation of the color of the view.

POWERING APPS WITH CODE

EXERCISE

KEY OBJECTIVE(S)

Make an app that adjusts a view's parameters with user interface elements and Swift code.

TIMING

30 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

Create a new app with two screens:
- One has a button, a text field, and a text label, when the button is pressed, make the label say: `"Hello, \(textFromTextField)!"`
- The second has two text fields, a button, and a text label. When the button is pressed, make the label print the sum of the ints in the text fields.

Bonus: Make the keyboards on the second screen numerical; display an error if the number is invalid. Perhaps change the .textColor to red for the label as well.

POWERING APPS WITH CODE

EXERCISE

KEY OBJECTIVE(S)

Create classes used as an app's object model.

TIMING

40 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

Create a Math class that can add two numbers, multiply two numbers, divide two numbers.

Create an app that contains two text fields and several buttons, representing math operations. Hook this up to your Math class.

DOCUMENTATION DEMO