CSC 581: Mobile App Development

Spring 2018

Unit 1: Getting Started with App Development

- Swift
  - language history, playgrounds vs. projects
  - let vs. var, types, type inference
  - operators, if-else, ~~switch~~, ~~ternary op~~
- SDK
  - XCode IDE, build+run+debug , documentation browser
  - interface builder, storyboards, object library
  - outlets vs. actions, attributes inspector

Swift notes

- Apple released Swift in 2014, for iOS and macOS development
  - easier to use & understand alternative to Objective-C
  - now also used for watchOS and tvOS development
  - Swift has been open-source since 2015, with a large community of supporters
- Swift is fully object-oriented, but has the look of cleaner scripting languages
- XCode is Apple's general-purpose IDE
  - includes playgrounds – files that serve as interactive Swift interpreters
  - can type Swift code in a playground & immediately see the results
  - the default playground includes `import UIKit` for loading standard UI elements

- in Swift, it is good practice to distinguish between constants and variables
  - constants are declared using `let` e.g., `let max = 100`
  - variables are declared using `var` e.g., `var num = 5`
- naming conventions for constants/variable are more liberal than Java
  - no spaces or math symbols, can't start with a digit
  - all legal: `maxValue` `salary$` `mañana` `π` 
- when printing values, can embed variables in text using \( )
  - e.g., `print("The number is \(num)")`

Swift notes (cont.)

- Swift is type-safe, the type of every value must be known by the compiler
  - you can declare the type along with a constant/variable
    `var age: Int` (primitives are `Int`, `Double`, `String`, `Bool`)
  - this is rarely done, as Swift performs type inferencing
    `var age = 21` (automatically infers that age is of type `Int`)
- Swift provides same math operators as Java
    `+, -, *, /, %, +=, -=, *=, /=, %=`
  - note: no `++` or `--`
  - unlike Java, operators must be applied to values of same type

```
var sum = 2 + 3.5              // illegal
var sum = Double(2) + 3.5      // but can cast values
```

- in Swift, if-else is similar to Java, but the test doesn't require parentheses
  - Boolean operators are the same: `==, !=, <, >, <=, >=, &&, ||, !`

```
if x > 0 {
    print("\(x) is positive")
}
```

- Unit 1 also covers switches and the ternary operator – I WILL NEVER USE THESE

3

SDK notes

- XCode can be used to create/open a playground or a project
  - when you create a project, select Single View App
- the XCode workspace is divided into 5 areas:
  1. Editor Area (center): where you edit Swift files and UIs
  2. Toolbar (top): buttons for build/run, controlling windows, etc.
  3. Navigator (left): default is Project Navigator, which shows your project files
  4. Debug (bottom): includes console pane, which shows results of `print`
  5. Utility (right): context-sensitive, includes attributes editor, UI library
- projects can be run directly on a device (must set up an Apple Dev. account)
  - we will mostly use the simulator, which can simulate any iOS device
  - XCode has detailed error messages & warnings, an integrated debugger
- the Single View App template automatically creates several files (in Navigator Area)
  - `ViewController.swift`: contains the Swift code that controls the app
    - ✓ clicking on this file opens the source code in the Editor Area
    - ✓ `ViewController` is a class with methods for controlling the app
    - ✓ option-clicking on a code element brings up the documentation browser
  - `Main.storyboard`: contains the UI for the app
    - ✓ clicking on this file opens Interface Builder, an integrated tool for drag & drop UIs

4

SDK notes (cont.)

- within Interface Builder, can drag UI elements from Object Library (in Utility Area)
  - e.g., can drag button element into screen, align using guide lines
- once an element is placed in the screen, can view/edit its attributes
  - select the element, click on Attributes Inspector icon at top of Utility Area
    e.g., can change button text, color, …
  - can place IB and the Editor side-by-side by clicking the linked-circles icon at top-right

- creating an action (i.e., associating a method in `ViewController` with a UI element)
  - right-click (or Control-click) on the element and drag the line inside the class
  - select `Connection:Action` and give specifics to create a method
    e.g., `Name:buttonClick`, `Type:UIButton` creates method named `buttonClick`
  - can then add Swift code to the method to specify the action

- creating an outlet (i.e., adding a reference to a UI element in `ViewController`)
  - right-click (or Control-click) on the element and drag the line inside the class
  - select `Connection:Outlet` and give specifics to create a field
    e.g., `Name:myButton`, `Type:UIButton` creates field named `myButton`
  - can then refer to that field in methods

5