University of Regina

CS 455/855
Mobile Computing

Xcode & iOS Programming

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs455/2018F

Readings

- Textbook, Chapters 6-9

- iOS Developer Library
  - Start Developing iOS Apps (Swift)

iOS Programming

- A significant part of iOS programming is making effective use of the functionality that is provided "for free".
- The biggest challenge is choosing the right API for your needs.
- Sometimes, the hardest part is knowing where to look for the functionality you require.
  - Apple Developer Library
  - Other tutorials
  - Within the IDE: Xcode

Xcode

- Xcode is the Integrated Development Environment (IDE) for iOS app development.
- It supports the full range of app development tasks:
  - create a new project
  - design the interface
  - write your code
  - compile & debug
  - view documentation
  - run the app on the simulator
  - access your developer credentials to sign the code
  - run the app on your device
  - evaluate the performance of the app

Programming in Xcode

- Create a project
  - Various templates are provided
- Design the user interface
  - Interface Builder lets you design the interface graphically & save it as resource files
- Write code
  - There are many features that allow you to write code quickly and easily
- Build & run the application
  - The iPhone Simulator allows testing on a computer
- Measure & tune application performance
  - Instruments for dynamic performance analysis

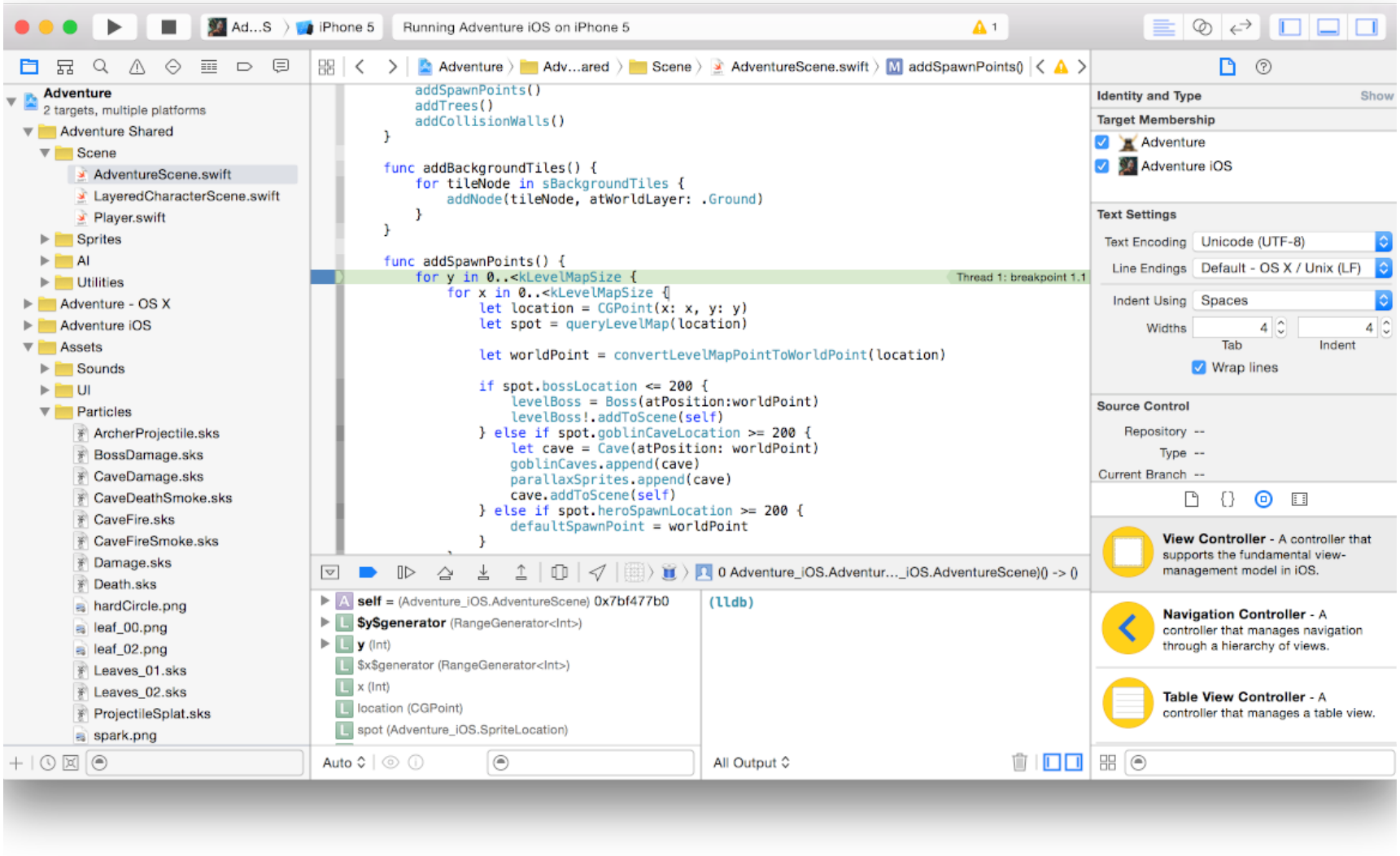
Application Templates

- Single View
  - single view, storyboard, and view control manager
- Master-Detail
  - navigation controller for a list of items and split-view on the iPad
- Page-Based
  - multi-page framework and a page view controller
- Tabbed
  - tab bar controller and tab bar items
- Game
  - support for graphics and animation
- Document Based
  - focus on document creation and storage
- Augmented Reality
  - video and graphics support for AR

Creating a New Project

- When you create a new project, a series of dialog screens will be shown to help you customize the templates:
  - application type
  - product name
  - team/organization name
    - leave this as it is for now; once you have a developer account, you can change it to “University of Regina (Department of Computer Science)”
  - organization name
    - your name
  - organization identifier
    - make this ca.uregina.cs.[“455” | “855”].[username]
    - mine is ca.uregina.cs.455.hoeber
  - language (Swift | Objective-C)
  - other options

Xcode Project Window



Features of the Text Editor

- Header-file lookup:
  - view the header file that declares the symbol by holding down the **command key** and clicking on a highlighted item
- API reference lookup:
  - get API reference about a symbol's usage by holding down the **option key** and clicking on a highlighted item
- Code folding
  - collapse code that you're not working on by clicking in the gutter (must be enabled in the editor options)
- Code completion:
  - As you type the name of a symbol, Xcode recognizes it and offers a suggestion
    - accept: Tab or Return
    - display options: Escape

Interface Builder (IB)

- IB is a visual design tool for creating interfaces for iOS and macOS applications.
- It allows you to assemble windows, views, controls, menus, and other elements from a library of configurable objects.
- You can:
  - arrange objects
  - set their attributes
  - establish connections between them
  - establish connections to your custom controller classes
  - configure the auto-layout parameters

Nibs and Storyboards

- In earlier versions of Xcode and iOS, you had to build each interface screen independently.
- The file type was .nib (NeXTStep Interface Builder)
- Now, we build all of the interface screens that are linked together in one file, of type .storyboard.

- Although you will see the “nib” terminology used, you will not need to build a stand-alone .nib file.
- If you find a book or tutorial talking about “Nib Loading”, you should look for a more current way of doing things.

5-minute Interface Builder Demo

- I will go through a demo that illustrates how easy it is to get started with building an app interface an linking it to software code.

Library Window

- The library window contains all the objects and resources that you can add to your IB documents
  - the library is context sensitive, only showing you the interface elements that are valid for the type of IB document you are building (iOS vs. macOS)
  - simply drag and drop
    - windows and menus can only be dropped at the top level
    - most others (views, input controls, etc.) can be dropped onto one another

Inspector Window

- The inspector window allows you to manipulate the settings for the currently selected object
  - the important settings for an interface element are:
    - attributes
    - size
    - connections
  - the details for each of these are dependent on the specific interface element selected in the main view

A Note on IBOutlets and IBActions

- Within a View Controller class, there are two hints in the code that say what IB can do
  - IBAction
    - says that a method can be used as the action for some event (this is an instance of the target-action design pattern)

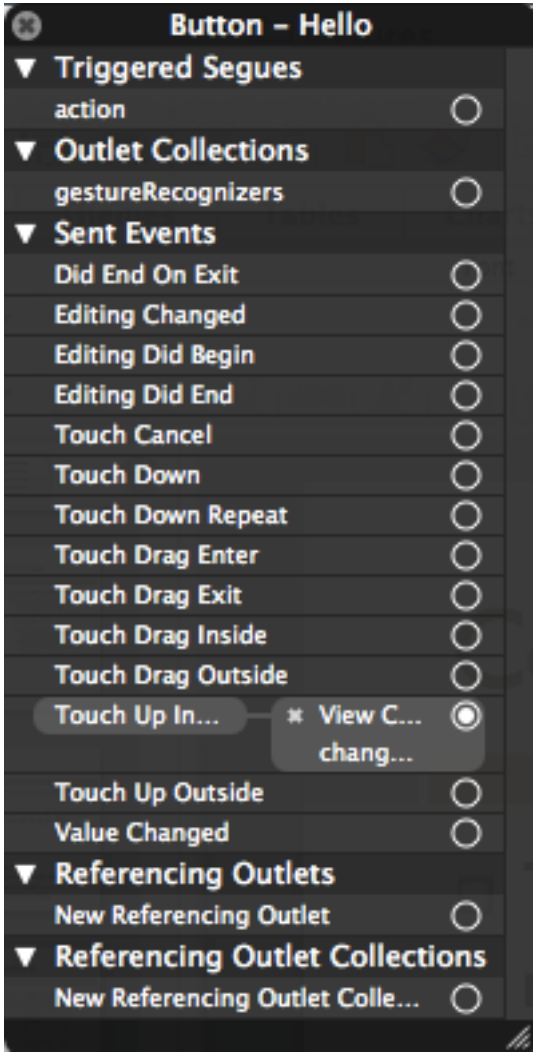
```
@IBAction func updateValue(_ sender: Any) {
}
```

  - IBOutlet
    - says that an instance variable can be linked to an interface element
    - used to access or assign values into the element

```
@IBOutlet weak var slider: UISlider!
```

Connections Panel

- The connections panel is a convenient way to manage and manipulate the connections between objects in the IB document
  - used to create connections for IBOutlets and IBActions
  - accessed by control-clicking any object



Using Interface Builder

- The easiest way to learn how to use Interface Builder is to:
  - read the documentation
  - create some simple applications
  - add additional controls to the interface
  - make things talk to one another
  - play around with the different event types
  - write code into your view controller class to make the interaction events do something interesting

Build & Run Applications

- The application can be built using either the device or the simulator
  - the simulator provides an environment that closely resembles an actual device
    - allows running the iOS applications in macOS for testing the core functionality
    - you can tell Xcode which device simulator to use
    - running applications in the iPhone simulator is not the same as running in actual devices
      - does not emulate processor performance nor the memory constraints
      - some hardware-based functionality is unavailable
  - running the application on a device requires provisioning profiles and certificates, which we'll get setup next week

Debug Applications

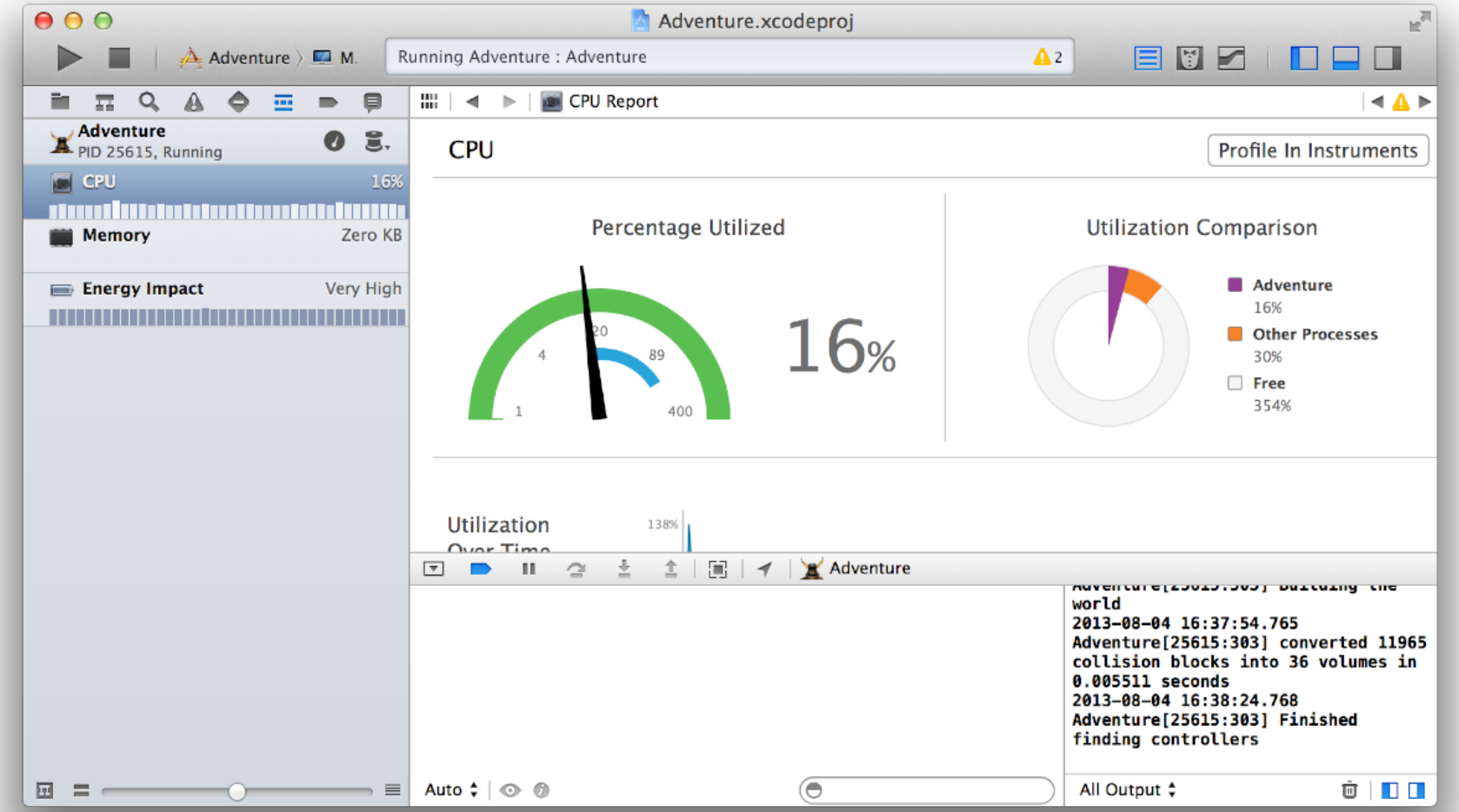
- When debugging you code, you can do the following
  - add and set breakpoints
  - view your call stack per thread
  - view the value of variables by hovering the mouse pointer over them
  - execute a single line of code
  - step in to, out of, or over a function call

- The debugger is used if there are any breakpoints in your code (added by clicking in the left gutter)
  - to delete a break point, drag it out of the gutter

Profile Your App

- Optimizing the performance is very important for software development on mobile devices
  - limited memory space
  - less CPU power
  - constraints on power usage
    - has direct impacts to battery-life
- You can profile your app while running it from Xcode by clicking on the Debug Navigator icon
- Or you can do detailed profiling by holding down the “play” icon

Debug Navigator



Build Configurations

- By default, you application is going to be compiled in debug mode
  - compiled line-by-line, without optimization
  - allows the debugger to work
- Before you ship your application, you should change the build to include the release mode
  - compiler optimization is enabled
  - takes longer to compile (sometimes much longer)
  - your code will run as efficiently as possible
- Select your project – select the Build Settings tab – Levels sub-tab – Build Active Architecture Only

Documentation

- The documentation for using the built-in iOS API (Cocoa classes) is included within Xcode.
- Different ways to access the documentation:
  - Window – Documentation and API Reference
  - Help – Documentation and API Reference
  - Command-Shift-0
  - Hold Option while hovering over a term in your code, and when it becomes underlined, Option-double-click
  - Quick Help in the Utilities bar
- Use the search feature of the documentation window.

Writing Your Documentation

- You should writing your own documentation with Xcode in mind.
- If you document your methods in a certain style, it will be available within Xcode when you want to use these methods.
- Quick Help
  - put your method documentation within a /** … */ comment style
  - paragraphs that begin with “Parameter xxx:” will put xxx in the parameter list
  - paragraphs that being with “Returns:” will put the information in the returns list
  - Formatting can be specified using Markdown

- Using //MARK: xxx will put a separator within the method navigator drop-down list
- Using //TODO: xxx or //FIXME: xxx will put the method that follows at the bottom of the navigator bar with the xxx note.

Version Control

- Xcode supports version control using git or subversion.
- By default, this will be activated, and the git repository will be on your workstation.
- If you have a git server, you can specify that the code be stored there.

- Why would you want to use version control in your software development practices?
  - security
  - collaboration
  - freedom from fear

Version Control Tips

- You can see which files you have modified (they have an M beside them in the file navigator.
- You can commit these modified files by right-clicking, and selecting Source Control – Commit
- You can roll-back to the previous version by right-clicking and selecting Source Control – Discard Changes
- If you want to see what has changed in the file, select the Version Editor (icon with two arrows pointing away from each other)
- If you are working in a team environment and you want to see who has changed what, long-click on the Version Editor button and select Blame.
- **Use version control for both your assignments and your project.**

Editor Configuration

- You can configure the editor to suit your specific needs (and screen size).
  - Xcode – Preferences – Fonts & Colours
  - Xcode – Preferences – Text Editing

- If your code indenting gets messed up, you can fix it:
  - select the code, and click Control-I
- There are some general patterns for code that are provided as snippets in Xcode
  - accessed using the Code Snippet Library in the Utilities bar (icon looks like {}, scroll down to the Swift ones)

Live Syntax Checking

- The live syntax checking can be very useful at times, and very annoying at other times.
  - For example, as you are typing code and pause to think, it will tell you that there is a syntax error. Duh.
  - This can be turned off in the General tab of the Preferences.
  - I find that I alternate between having it on and having it off (on until it bugs me, then off until I need it again).

Debugging

- Everyone has their own personal style for debugging code.
- There are two methods that are common:
  - caveman debugging
    - dumping messages to the console
    - use print or the os.log object
    - may also put things in the setter observers of variables
    - be sure to remove these when you have solved the problem
  - breakpoints, tracing, & inspection debugging
    - Xcode provides an excellent set of tools for adding breakpoints, tracing through the code, and inspecting variables
    - There are some great features available for controlling what happens when the break is encountered (control-click the breakpoint, and then Edit Breakpoint)

Running on a Device

- Running an app under development on a device used to be a big pain.
- Things are much simpler now, but there are still some details to attend to.

- The key issue is that an app needs to be *signed* before it will run on a device.
- This is a security measure to avoid malicious apps from becoming commonplace.

- Apple Developer Program
  - needed to release an app
  - not needed for installing an app for testing purposes only
  - we will use the University Developer Program

University Developer Program

- An Apple University Developer Program ID will be created for you:
  - it is your URegina email address
  - you will received an email about this soon, and will need to create an account
  - enter this into Xcode
    - Xcode – Preferences – Accounts (click the + at the bottom to add it to the list)
  - Your team is University of Regina (Department of Computer Science)
  - If you have already created an app, you should select this team in the Preferences for the project (General tab)

Automatic Development Signing

- New in the latest version of Xcode, the signing of the code happens automatically for you if you have your Developer Program and Team information entered correctly.

- When you connect your device to the computer, it will appear in the build list beside the project name (e.g., where you select which device to use in the simulator).
- It will probably say the device is not available the first time. Select it anyway, and then you will have the ability to register the device. Once this is done, you will be able to run the app on the device.
- Once the app is there, it remains available even when the device is disconnected from the computer.

Homework

- Make sure you have read Chapters 1 – 5 (Swift) and Chapters 6 – 9 (Xcode)

- Start reading the "App Programming Guide for iOS"

- Assignment #1
  - due Oct 5
- Short Paper #1 (CS 855)
  - due Oct 12