CSC 581: Mobile App Development

Spring 2018

Unit 3: Optionals & segues

- Swift
  - optionals, guards
  - scope, enumerations
- UIKit
  - segues
  - navigation controllers, tab bar controllers
  - view controller life cycle
  - navigation hierarchies & design guidelines

Optionals

recall: Swift uses optionals when a value may or may not be present

```
var ages = ["Pat": 18, "Chris": 20, "Kelly": 19]
print(ages["Chris"])              → Optional(20)

outputLabel.text = "Hello world"
print(outputLabel.text)           → Optional("Hello world")
```

- you can unwrap an optional using ! or (more elegantly) if-let

```
print(ages["Chris"]!)             → 20

if let msg = outputLabel.text {
    print(msg)                    → Hello world
}
```

every type in Swift has a corresponding optional type (with ? at end)

```
var age: Integer? = ages["Chris"]

var msg: String? = outputLabel.text
```

Intentional optionals

structs/classes can have optional fields; functions can return optionals

```
struct Profile {
    var name: String
    var age: Int?
    var favoriteMovie: String?

    func getAge() -> Int? {
        return age
    }
}

var person = Profile(name: "Sean", age: 22, favoriteMovie: "Life of Pi")
if let age = person.getAge() {
    print("\(person.name) is \(age) years old.")
}
else {
    print("\(person.name)'s age is unknown")
}

person = Profile(name: "Dave", age: nil,
                 favoriteMovie: "Big Trouble in Little China")
if let age = person.getAge() {
    print("\(person.name) is \(age) years old.")
}
else {
    print("\(person.name)'s age is unknown")
}
```

3

Guards

when defining functions, will often have an initial validity test or tests

```
func average(_ nums: [Double]) -> Double {
    if nums.count > 0 {
        var sum = 0.0
        for n in nums {
            sum += n
        }
        return sum/nums.count
    }
    else {
        return 0.0
    }
}


func display(for person: Profile) {
    if let age = person.age {
        if let fav = person.favoriteMovie {
            print("\(person.name) (age \(age)) loves \(fav)")
        }
    }
}
```

4

Guards

a guard is a cleaner notation for handling validity tests

```
guard condition else {  }
```

```
func average(_ nums: [Double]) -> Double {
    guard nums.count > 0 else { return 0.0 }

    var sum = 0.0
    for n in nums {
        sum += n
    }
    return sum/nums.count
    }
}
```

```
func display(for person: Profile) {
    guard let age = person.age, let fav = person.favoriteMovie else { return }

    print("\(person.name) (age \(age)) loves \(fav)")
}
```

5

Variable scope

as in most languages, a block (code enclosed in { }) defines a new scope for variables

- note: a function defines a new scope as well as control statements

```
var x = 100
if true {
    var y = 3
    print("\(x) \(y)")      → 100 3
}
print(x)                    → 100
print(y)                    → ERROR: Use of unresolved identifier 'y'
```

variable shadowing

- unlike Java, you can override an existing variable inside a new scope

```
var x = 100
if true {
    var x = 3
    print(x)                → 3
}
print(x)                    → 100
```

6

Enumerations

as in Java, you can define a new type by enumerating all of the possible values of that type

- useful when you want a limited range of values to be assignable

```
enum Answer {
    case yes, no, maybe
}

enum DayOfWeek {
    case Monday, Tuesday, Wednesday, Thursday,
         Friday, Saturday, Sunday
}

var response = Answer.yes

if (response == .yes || response == .no) {    // Swift infers the
    print("That's a decisive answer")         // enum type
}

var today = DayOfWeek.Tuesday
print(today)
```

7

Segues

a view controller manages a screen within an app

- if an app is to have multiple screens
  - ✓ need a view controller for each screen
  - ✓ need to define how to transition from one view controller to another

a segue defines a transition between view controllers

- segues are defined in Interface Builder by connecting the controllers
- can select the presentation method for the transition
- navigation controllers allow you to go back or jump to other screens

complete Lesson 3.6

- create an app with multiple views, define segues between them
- final version has two destination buttons, an enable/disable switch

HW3a: complete the lab at the end of 3.6

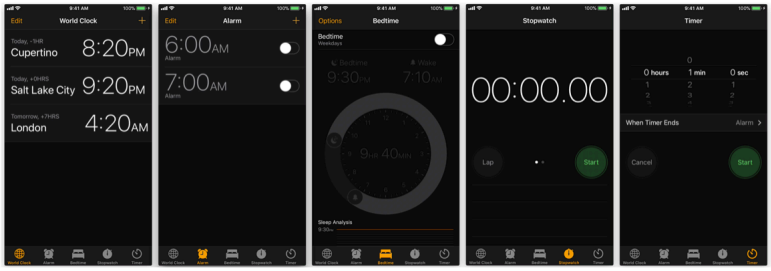
- create an app with a login screen

8

Tab bar controllers

a tab bar allows you to arrange your app into distinct sections

- e.g., Apple Clock app: World Clock, Alarm, Bedtime, Stopwatch, Timer



a tab bar is defined using a Tab Bar View

- shows tabs (icons) across the bottom of the screen
- each tab has its own navigation hierarchy
- the tab bar controller coordinates the navigation

9

Adding a tab bar

first, create the main View for your project

- select that View in IB, then choose Editor > Embed in > Tab Bar Controller
- this adds a Tab Bar Controller to the canvas as well as adding a UITabBarItem to the bottom of the View

to add another tab bar item

- drag a View Controller from the Object Library to the canvas
- connect it by control-dragging from the Tab Bar Controller to the new View Controller, and choosing "view controllers" under Relationship Segue
- this adds a second tab bar item to the Tab Bar Controller

customize the tab bar item

- select the desired tab bar item in a View
- in the Attributes Inspector, select the desired icon under System Item
  e.g., Favorites, Features, Search, Downloads, …
- you can change the title under Bar Item > Title
- you can select a custom icon image under Bar Item > Image

10

Adding behavior to the Views

by default, the first View has a ViewController.swift file already defined

- as we have done before, can add UI elements to the View and link to Outlets and Actions in the ViewController

to add ViewControllers for the new View

- select File > New > File… > Cocoa Touch Class
- set the subclass of the new class to UIViewController and give it a unique name
- connect the View and the new ViewController by selecting the ViewController in the Document Outline (left side of IB)
- then, in the Identity Inspector (icon to left of Attributes Inspector), set the custom class to the new Swift file

HW3b: complete the guided project (Personality Quiz) at the end of Unit 3

- substitute your own questions and quiz results

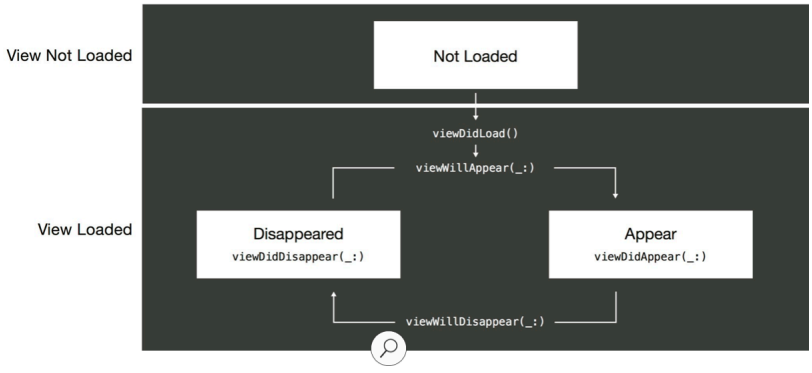
11

View Controller life cycle

view controllers can be in one of 5 states

1. not loaded
2. appearing
3. appeared
4. disappearing
5. disappeared

as a view transitions from one state to another, iOS calls a built-in method



12

View event management

viewDidLoad()

- performs tasks that need to be doneonce, when app is first loaded
  e.g, update a label's font, size, or color

viewWillAppear()

- performs tasks that need to be done each time the view is to appear on the screen
  e.g, refreshing views, adjusting to new orientation, accessing location

viewDidAppear()

- waits until the view is fully loaded – better for complex or slow tasks
  e.g., starting an animation, fetching data

viewWillDisappear()

- performs tasks that need to be done when the user navigates away from the screen
  e.g, refreshing views, adjusting to new orientation, accessing location

viewDidDisappear()

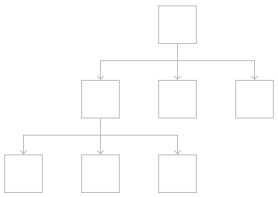
- waits until the user has navigated to a new view
  e.g., stop services related to the old view (such as background audio)

13

Navigation hierarchies

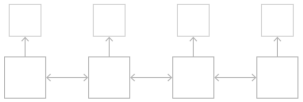
hierarchical

- user makes one choice per day until reaching a destination
  e.g., Settings, Mail



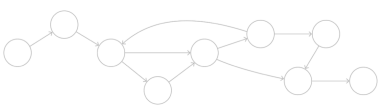
flat

- user switches between multiple content categories (e.g., with tab bar)
  e.g., Music, App Store



content- or experience-driven

- the content defines the navigation
  e.g., games, books, other immersive apps



14

Navigation design guidelines

Design an information structure that makes it fast and easy to get to content.

- Organize your information in a way that requires a minimum number of taps, swipes, and screens.

Use standard navigation components.

- Whenever possible, use standard navigation controls, such as tab bars, segmented controls, table views, collection views, and split views. Users are already familiar with these controls and will intuitively know how to get around in your app.

Use a navigation bar to traverse a hierarchy of data.

- The navigation bar's title can display the user's current position in the hierarchy, and the Back button makes it easy to return to the previous position.

Use a tab bar to present peer categories of content or functionality.

- A tab bar lets people quickly and easily switch between categories or modes of operation, regardless of their current location.

Use the proper transition style.

- Whenever you're transitioning to a new screen to display more detail about an item, you should use a right-to-left push transition used by a Show segue within a navigation controller. If the user is switching contexts - from displaying contacts to adding a new contact, for example - use a modal transition.

15