Lesson 6

PDF Slides Lesson 6

Lesson6_slides.pdf https://drive.google.com/file/d/17COY6eokcLOPyEPI3O0wHUALYfUKxV8h/view?usp=drivesdk

Scenes

- What are Scenes?
  - We know from Lesson 5 that View Controllers define a scene in our app.
  - A scene is simply ‘one thing’ we can do in our app that is described by ‘one screen’.
  - Complex apps will require multiple scenes to be able to deliver the required functionality.
  - Consider social networking apps – inbox, feed, etc.

Segues

- To move between scenes, we need to define the segues between the View Controllers. A segue defines how a new **ViewController** appears
  - over the previous **ViewController**.
  - We can also use segues to dismiss ViewControllers to move back to previous ones.
  - These are termed modal segues.

Triggers

- Segues can be trigged either using the Interface Builder or programmatically using Swift code.

- We generally connect the trigger to a control such as a button, so the segue is triggered when the user taps the button.
- We will go through how to trigger a segue both programmatically and using Interface Builder in the exercises.

Unwinding

- Dismissing a segue is termed unwinding it.
  - We can't really do this using Interface Builder.
  - We do this programmatically by creating an **IBAction** function that takes a **UIStoryboardSegue** parameter.
  - We then need to connect the **IBAction** to a control, which we will do in the exercises.

Navigation Controllers

- Presenting some tasks suits a modal segue (such as composing an email in an email
  client).
  - This will allow the user to return to the previous **ViewController** when finished.
- Other styles of tasks require segues from one View Controller to another.
  - Consider selecting a folder in an email client.
- Using a Navigation Controller we can push a new **ViewController** on top of the stack with a push transition.
  - When we are done, we can click 'Back' to dismiss (or pop) the view and return to the next view on the stack.
  - This can be many, many levels deep (although too many would probably indicate a level of complexity in the App that is not very suitable).

Navigation Bars and Items

- We all know the Navigation Bar at the top of the screen from Lesson 5 (and likely
  experience).

    - This is where there is a Title and sometimes some buttons as well.
    - This can be customized visually in Interface Builder, keeping in mind the Human Interface guidelines
    - The Navigation Bar is implemented as a **UINavigationBar**.
- To customize what the bar says, we need to modify the Navigation Item within our View Controller.
    - Where we can actually change the name...
    - ... and also what the 'Back' button says.
- We can present the 'new style' large titles by modifying the attributes of the Navigation Bar.

Passing Information

- Generally, we will need to pass information between View Controllers.
    - This is despite changing View Controllers to a different 'task', as it is still part of a larger workflow.
- Consider the Contacts app, where we tap on a contact's name (such as David McMeekin).
    - How does the App know to bring up his details?
- We override the prepare function to set up data for transfer during a segue.
- The prepare function is how we programmatically define a segue!

Other View Controllers

- A **NavigationController** or modal segue isn't the only way to move between View Controllers.

Tab Bar Controllers

Adding a Tab Bar Controller

- Tab Bar Controllers are commonly seen in many apps to separate different workflows within an App.

    - Consider the social networking example again – there's that bar at the bottom in Instagram, Facebook, Apple Music, Spotify...
- To add a Tab Bar Controller in Interface Builder, we drag it onto the Canvas from the Object library.

Adding Tabs

- To add a tab, we drag it from the Object Library into our **TabBarController**.
    - We can adjust the attributes to adjust the look and feel.
- Each of these tabs is termed a Tab Bar Item.

Tab Bar Items

- Each Tab Bar Item has a Label and an Image (Icon).
    - We can use a bunch of built-in icons or supply our own.
    - Favorites, Downloads, Search, Contacts, and many more...
- We can customize the look and feel further using Swift code.
    - We can customize any element with Swift, however, some require us to, such as the (red) Badge.
    - We often use the Badge to bring attention to a tab, for new content or content requiring action (e.g. unread messages).

Navigation Hierarchy

Why?

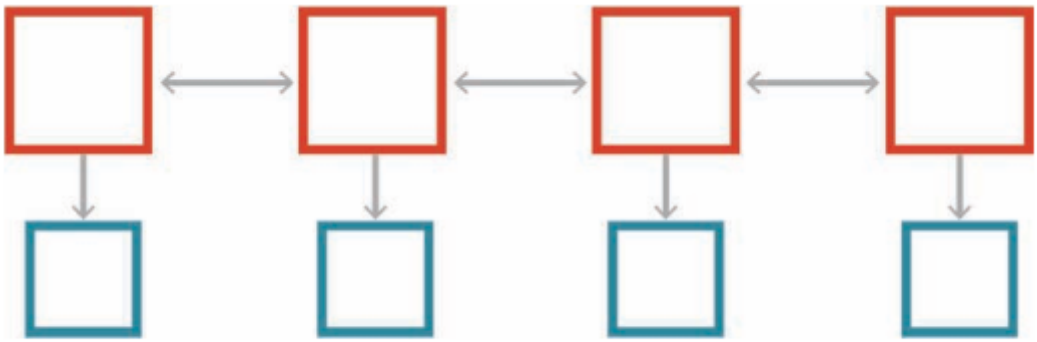
- We are familiar with the Human Interface Guidelines for the look and feel of an app.
    - These ensure a consistent user experience between apps, such that users can infer how to use new apps based on how they used existing ones.
- This aim is met also through the design of the Navigation Hierarchy.
    - This is namely how a User moves between View Controllers to solve tasks within the App.

Hierarchical Navigation

- A user makes one choice per screen/scene until they reach their destination scene.
  - If they change the destination, they have to start from the beginning, retracing their steps back to the start.
  - Used in apps such as Settings.
- Generally implemented with Navigation Controllers.

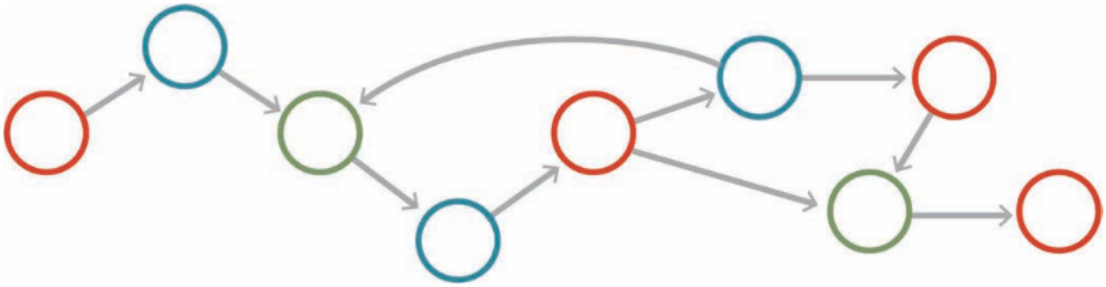
Flat Navigation

- Users can switch between different categories or 'workflows'.
  - Consider Facebook, Spotify, Instagram, or similar apps.
- Generally implemented with a Tab Bar Controller.
- Sometimes Apps can use both styles of navigation and hence implement both types of controllers.



Content-Driven Navigation

- This is a different and more free-flow type of navigation, beyond the scope of this course.
- Users can move in a non-linear fashion between elements of the App.
  - The most common example would be games



Workflows

- It is best to consider what the User will be doing in the App before building it.
  - This is to be able to best design the navigation hierarchy.
  - Consider the features and how they will be built as View Controllers, as well as the movement between them.

Design Guidelines

- Apple recommends the following things to consider when designing a Navigation Hierarchy.
  - Design a structure to make it fast and easy to get to the content.
  - Use standard navigation controls, such as a Navigation Bar when implementing a Hierarchical Navigation.
  - Use a Tab Bar when implementing multiple categories (modes, workflows) of content or functionality.
  - Use the correct style (modal vs push) where appropriate!

ViewController Even Handling

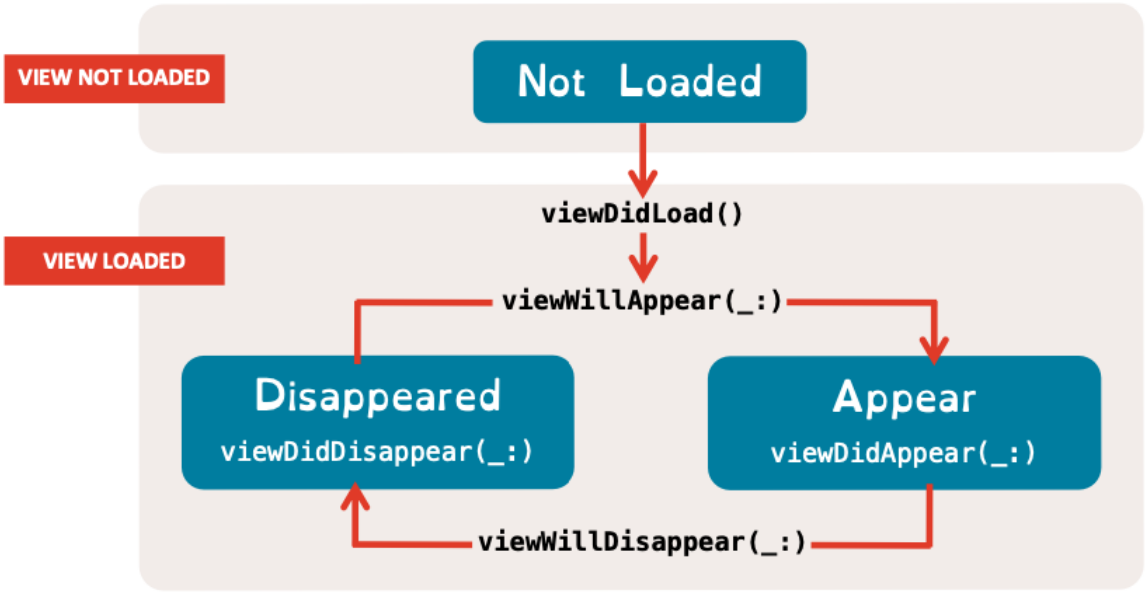
Life Cycle

- When considering the logic behind our App's structure, the view structure is an essential component.
  - When does each View Controller load?
  - What should it do when it does?

- What about when it disappears?

Life Cycle States

- A View Controller can be in one of the following states:
  - View not loaded
  - View appearing
  - View appeared
  - View disappearing
  - View disappeared



View Did Load

- The most common function is **viewDidLoad**, which runs when the View Controller is loaded.
  - In other words, when it 'appears' within the App.
- Generally, this function is used to programmatically initialize the View Controller.
  - Set up values, connect to resources, etc...

Other Event Handlers

- The other event handlers are generally used for similar purposes, such as ‘cleaning up’ or closing connections on exit.

Overriding

- One thing to be aware of is that you must call the superclass initializer within them as we are overloading each of these functions when you do so.

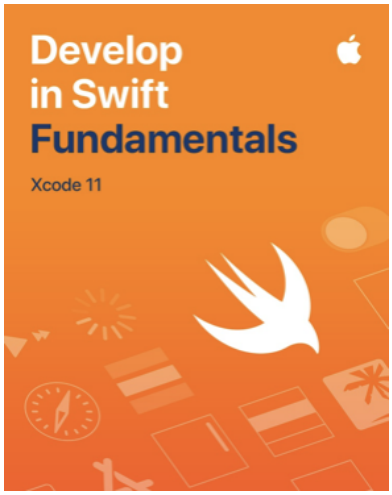
```
override func viewWillAppear(_animated: Bool) {
  super.viewWillAppear(animated)
}
```

Did vs Will

- Functions with “**will**” run before the event, whereas functions with “**did**” run afterward.
- **viewDidLoad** runs each time the View Controller is loaded, whereas **viewWillAppear** runs each time the view appears.

Extra Resources

- In Apple Books:



- 3.6 Segues and Navigation Controllers
- 3.7 Tab bar Controllers
- 3.8 View Controller Life Cycle