Lesson 7

PDF Slides Lesson 7

Lesson7_slides.pdf https://drive.google.com/file/d/17DpSo8pNWoG1mRWqcuM4KeuFPqDNluOk/view?usp=drivesdk

Protocols

- Swift is a protocol oriented language
- According to Swift documentation, protocols in swift "defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality"
- This Section would be looking at:
    - `CustomStringConvertible`
    - `Equatable`
- Types that conform to a protocol must implement all the requirements of the protocol

`CustomStringConvertible` Protocol

- A protocol to display how ojects are printed to the console
- A textual representation of the values contained in an object's variables
- Example of class without `CustomStringConvertible` Protocol:

```
import UIKit

class Book {
  var title: String
  var author: String
  var isbn: Int

  init(title: String, author: String, isbn: Int{
    self.title = title
    self.author = author
    self.isbn = isbn
  }
}

var book = Book(title: "Adventures", author: "Alex L", isbn: 100)
print("Book is \(book)")
```

- Using a `CustomStringConvertible`

```
import UIKit

class Book {
  var title: String
  var author: String
  var isbn: Int

  init(title: String, author: String, isbn: Int{
    self.title = title
    self.author = author
    self.isbn = isbn
  }

  var description: String {
    return "Book title: \(title), author: \(author), ISBN: \(isbn)"
  }
}

var book = Book(title: "Adventures", author: "Alex L", isbn: 100)
print("Book is \(book)")
```

`Equatable` Protocols

- Comparing two integers for equality is a simple `if x == y`
- Comparing objects can be more challenging
- The `Equatable` protocol allows objects to be compared for equality
- Example with `Equatable`:

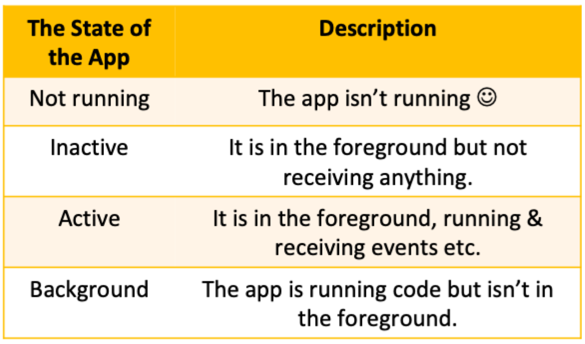
```
import UIKit

class Book {
  var title: String
  var author: String
  var isbn: Int

  init(title: String, author: String, isbn: Int{
    self.title = title
    self.author = author
    self.isbn = isbn
  }

  var description: String {
    return "Book title: \(title), author: \(author), ISBN: \(isbn)"
  }

  static func == (bookOne: Book, bookTwo: Book) -> Bool {
    let booksAreTheSame = bookOne.title == bookTwo. title &&
      booOne.author == bookTwo.author &&
      bookOne.isbn == bookTwo.isbn
    return booksAreTheSame
}

var book1 = Book(title: "Adventures", author: "Alex L", isbn: 100)
var book2 = Book(title: "Adventures", author: "Alex L", isbn: 100)
var book3 = Book(title: "Castles", author: "Alex L", isbn: 1020)

if book1 == book2{
  print("Same Book")
}
else {
  print(Different Book")
}
```

Delegation

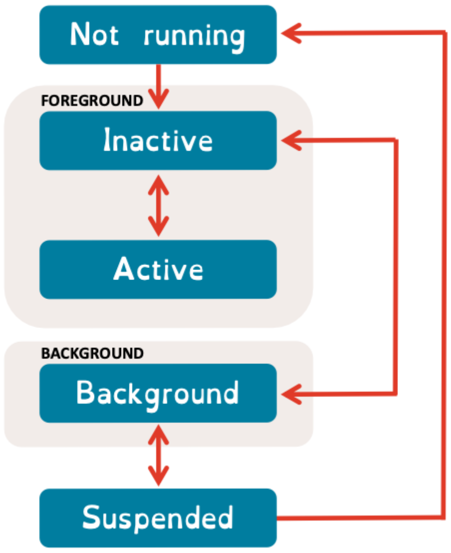
- Delegation is a design pattern where a class or structure delegates respoinsibility to another
- As with humans, there are two parties involved:
  - The one who delegates
  - The one who it was delegated to

Life Cycle

- 4 Stages of an App(Scene)'s Life



The State of the App	Description
Not running	The app isn't running ☺
Inactive	It is in the foreground but not receiving anything.
Active	It is in the foreground, running & receiving events etc.
Background	The app is running code but isn't in the foreground.



The App Delegate

- Previously, each app could only have one instance
    - As of IOS13, Scenes allow multiple instances (windows)
- Each Scene is an independent iinstance that can be in the foreground or background
- The App Delegate, within `AppDelegate.swift` , manages t he creation of the App as well as creating and destroying Scenes

`AppDelegate` Methods

- `application (didFinishLaunchingWithOptions)`
    - Run when the App has been opened, setup for all scenes
- `application(configurationForConnecting) -> UISceneConfiguration`
    - Run when a new Scene is created
- `application(didDiscardSceneSession)`
    - Run after a Scene is closed

The Scene Delegation

- Manages the lifecycle of each individual Scene
- need to manage the transitions of each scene between each state, foreground and background

- Located within `SceneDelegate.swift`

`SceneDelegate` Methods

- `scene(willConnectTo, options)`
    - When the scene is first created
- `sceneWillEnterForeground()`
    - When the Scene enters the foreground. Run before the Scene is Active but is in the foreground
- `sceneDidBecomeActive()`
    - Run after the scene is active in the foreground
- `sceneWillResignActive()`
    - When the Scene becomes inactive on the way to the background
- `sceneDidEnterBackground()`
    - When the scene moves into the background from the foreground
- `sceneDidDisconnect()`
    - When the scene is finished/closed/terminated

Model View Controller

- Model View is an architecture design pattern
- It Separates:
    - The Model of the data from
    - The view the user sees from
    - The control of what happens and where thigns go
- Apple traditionally uses this for their development
- Java, C#, Python, Ruby, PHP all use MVC frameworks
- Every object belongs to either hte Model, the veiw or the controller

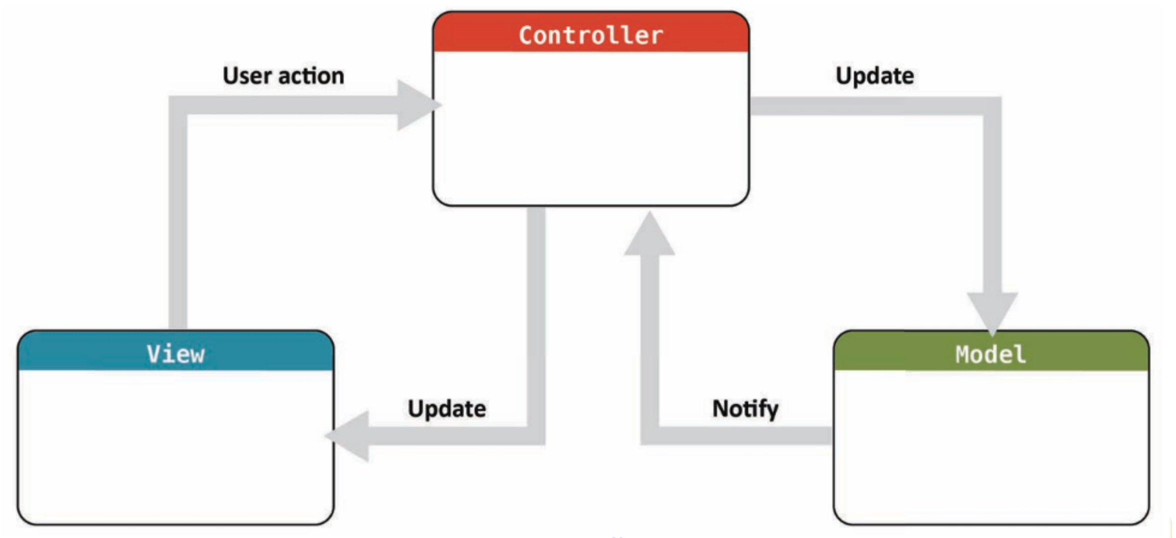
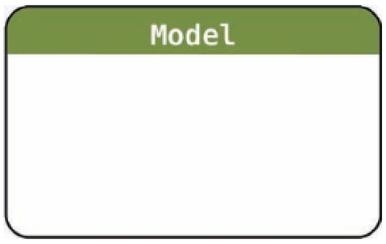


Image adapted from: https://developer.apple.com

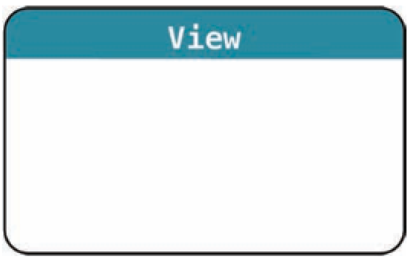
A Model

- Defines the structure of some data: group the data for seomthing specific into a Model
- Often relates to other Model objects
- Knows nothing about the view
- Models are often classes or structures
- The arrays/dictionaries created to store the colors in the app were also Models



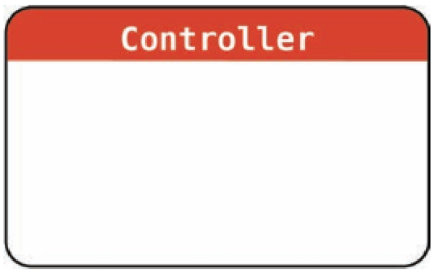
A View

- Views are objects seen by the user, the User Interface: buttons, labels, etc...
- Can draw itself on screen
- Can respond to user input
- Displays the data from the Model to hte user
- Allows the user to interact with App's data



A Controller

- Manages the App
- Configures the views for users
- Controls the View and Model to ensure correct data is displayed
- Is a message carrier between the Views and the Model(s)
- View, Model and Helper Controllers

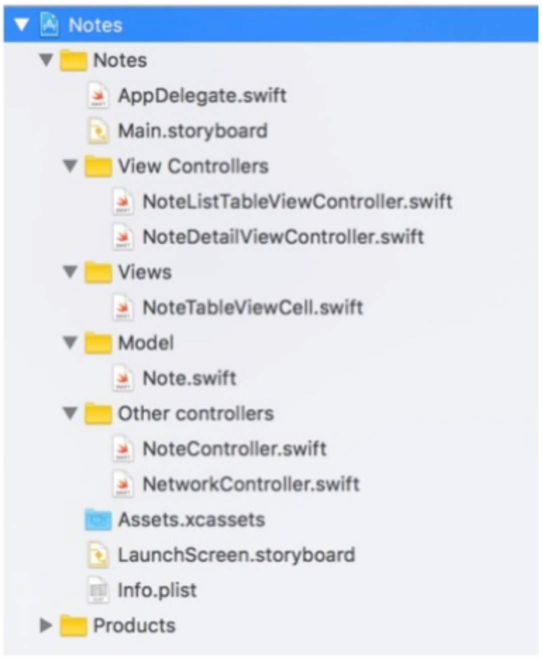


Projects in Xcode

- Xcode project can have lots of files:
    - Views
    - Storyboards
    - Structures
    - Classes
    - Protocols
    - Controllers

Organizing Projects

- File names must be descriptive
- Individual files for each type (classes, structures)
- Possible file structures
    - View controllers
    - Views
    - Model
    - Other



```
▼ Notes
  ▼ Notes
      AppDelegate.swift
      Main.storyboard
    ▼ View Controllers
        NoteListTableViewController.swift
        NoteDetailViewController.swift
    ▼ Views
        NoteTableViewCell.swift
    ▼ Model
        Note.swift
    ▼ Other controllers
        NoteController.swift
        NetworkController.swift
      Assets.xcassets
      LaunchScreen.storyboard
      Info.plist
  ▶ Products
```

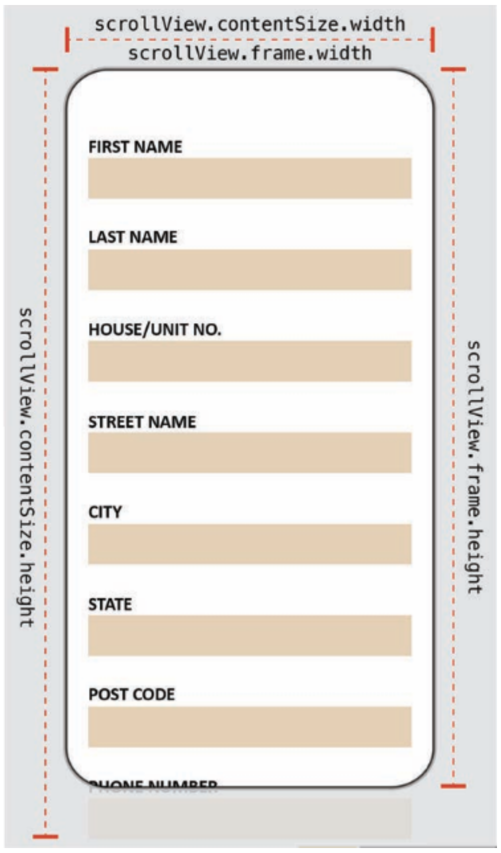
Scroll View

- Used when what needs to be displayed is larger than the screen the App is running on
- The class used is `UIScrollView`
- It only scrolls if content's size is greater than the frame's width or height

`UIScrollView` Class

- It requires 2 pieces of information:
  - Position & size of the scroll view
    - stored in the frame Property
  - The size of teh content to be displayed
    - Stored in the contentSize property
- These properties can be managed through `AutoLayout` and Interface

Builder



Family of Scroll Views

- Within the `UIKit`, `UIScrollView` has many child classes
- Two classes are:
  - `UITableView`
  - `UICollectionView`
- All of UIScrollView's functionality exists within `UITableView` and `UICollectionView`

Scroll Views

- Scroll views can be impletmented using Interface builder's Auto Layout feature
- Constraints can be added to the scroll view edges securing them to the view controller's view
  - Ensures the scroll view and view controller's view are the same size

- No matter what device it is displayed on, the scroll view and view controller's view are the same size

Table Views

- Table views are probably the most widely used view in IOS Apps
- It is a UITableView class
- Large amounts of data can be simply and beautifully displayed to the user
- Users can navigate through hierarchically structured data
- An indexed list of items can be presented
- Detailed information and controls can be displayed in visually distinct groupings and a selectable list of options can be presetnted

Table View's make up

- To facilitate large amounts of scrolling, single column list is provided
- The table view can have a header and footer
- Each row can be divided into sections or groups
- Each section can have
    - A header above the first item
    - A footer below the last item

Controllers

- Adding a table view to a project can be done in two ways:
    - Add a table view instance directly to a view controller's view
    - Add a table view controller to the storyboard
- The table view controller is a view controller subclass responsible to manage a single table view instance
- Table View controller is the data source and delegate of the table view
- These Already contain a large amount of functionality so it's not required to code it

    - The keyboard covers a text field within the table view, the view will always scroll
- The benifits of Table View Controllers means most IOS developers use them

Table View Style - Plain

- The default style
- Rows separated into labeled sections
- Optional index along the right edge of the table
- Sections follow on immediately, no spacing, an unbroken list is created

Table View Style - Grouped

- Visually distinct groups or sections with i spacing between is possible when displaying rows
- Index along the edge is not possible

Table View Editing

- Table views can be set to editing mode
- Users can then:
    - Insert new cells
    - Delete cells
    - Reorder cells

Table View Cells

- Every table row is represented with a table view cell: `UITableViewCell`
- Cells are reusable views and can display:
    - Text
    - Images
    - Any other `UIView`

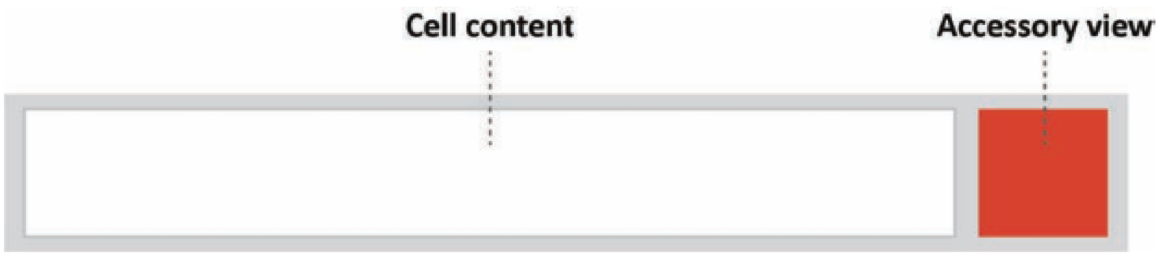


Image adapted from: https://developer.apple.com

- Each cell has an optional accessory view
- Cell content size shrinks in editing mode, allowing space for the editing and reorder controls

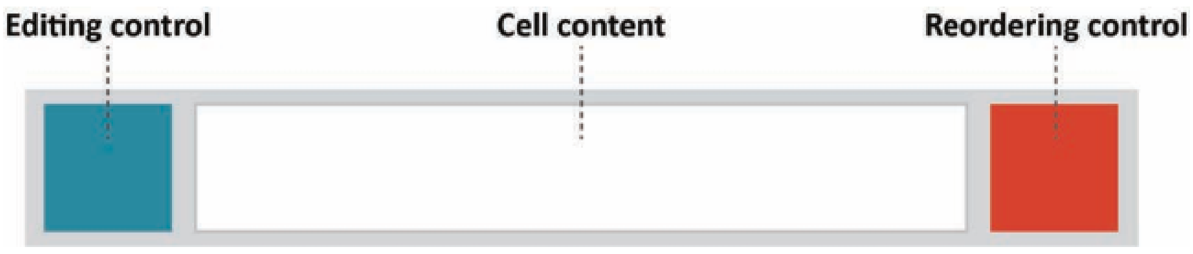


Image adapted from: https://developer.apple.com

- Three properties are defined for cell content:
    - `textLabel`, a `UILabel` for the title
    - `detailedTextLabel`, a `UILabel` for the subtitle
    - `imageView`, a `UIImageView` for an Image

Index Path

- Points to a specific row in a specific section of the Table View
- The row and section properties facilitate access
- They are zero based like arrays

Arrays

- Table views are fantastic in displaying similar data and are often undergirded with a collection of model objects
- An array is the usual choice although other are possible

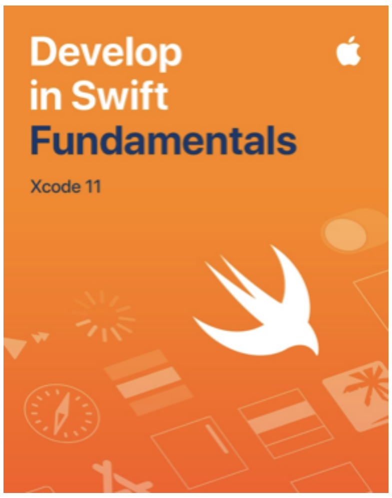
- An array has the count property, facilitating knowledge of how many pieces of data it has
- The table view's number of rows can be calculated from this

Dequeuing a Cell

- In displaying large data amounts, a table view could have large numbers of cells
- To prevent this Table View only load the visible cells plus a small number above and below what is visible
- Cells that leave the visible field can be reused on those things about to enter the visual field
- This is dequeuing - it uses the `reuseIdentifier` along with the `dequeueReusableCell()` method

Extra Resources

- In Apple Books:



- 4.1 Protocols
- 4.2 App Anatomy and Life Cycle
- 4.3 Modal View Controller
- 4.4 Scroll views
- 4.5 Table views

Source Code:

Ace5584/IOS-Dev-Notes

Contribute to Ace5584/IOS-Dev-Notes development by creating an account on GitHub.

https://github.com/Ace5584/IOS-Dev-Notes/tree/main