TP 7 - RSSReader

Adrien Humilière 03/05/2017

For this TP, you will work on the initial project provided on adhumi.fr/teaching.

Part 1

- Using the Project Navigator, observe how the project does not contain a storyboard or view controller class, but does include a main.swift file. This project is a non-traditional, "empty" iOS app project.
- Run the app and observe the black screen appear.
- Every Swift program has a starting point, or "application entry point," which is defined as the top-level code in a file called main.swift. Open main.swift and examine its contents.
- The top level code in main.swift just calls `UIApplicationMain`. `UIApplicationMain` is passed command-line arguments and the class name for the project's app delegate.
- Using the documentation, examine the description of the `UIApplicationMain` function. The `UIApplicationMain` function instantiates a `UIApplication` object, assigns the app delegate to the `UIApplication` object delegate property, begins the main event loop, and, if configured, loads the main storyboard interface.
- Delete the main.swift file.
- Run the app and use the Issue Navigator to observe the compilation error.
- Explain how Swift provides a `@UIApplicationMain` attribute that synthesizes a main entry point and eliminates the need for a main.swift file. Add the `@UIApplicationMain` attribute above the `AppDelegate` class definition.
- Run the app and observe the black screen appear.

Part 2

- Add a new storyboard called Main.storyboard, saving it in the Base.lproj directory, ensuring that the RSSReader group is selected, and ensuring that the RSSReader target is checked.
- Using the Project Navigator, select the RSSReader project and set the Main Interface attribute to Main.
- Run the app, and observe the console output. The app is reporting that a main storyboard is configured but has no default view controller.

- Select the storyboard and observe how the canvas is empty. Drag a new Tab Bar Controller into the canvas, and observe how Interface Builder includes two additional scenes with the tab bar controller. There is a warning indicating that although view controllers exist in the storyboard, the tab bar controller scene is unreachable because no initial view controller has been specified.
- Using Interface Builder, select the tab bar controller, open the Attributes Inspector, ensure that the Is Initial View Controller attribute is checked, and notice how Interface Builder displays an arrow to the left of the tab bar controller scene.
- Run the app, observe the default tab bar controller interface, and interact with the two tab buttons.

Part 3

- Observe the main storyboard, which displays the default tab bar controller scene and two default view controller scenes. The default tab bar controller consists of two tabs, with relationships bound to the two view controllers.
- Using the Project Navigator, explore the Images.xcassets asset catalog and observe the image sets, where icons have been provided for the project.
- Using Interface Builder, select the Item 1 tab bar button at the bottom of the Item 1 view controller, open the Attributes Inspector, and change the Title attribute to *Top Song*, the Image attribute to *Top Song Icon*, and observe how the canvas reflects the change.
- Select the Item 2 tab bar button at the bottom of the Item 2 view controller, open the Attributes Inspector, change the Title attribute to *Top Album*, change the Image attribute to *Top Album Icon*, and observe how the canvas reflects the change.
- Zoom out of the storyboard, and drag a new View Controller onto the storyboard. Use the Document Outline to rename it to Top App.
- Using the Document Outline, Control-drag a connection from the Tab Bar Controller to the Top App controller, select the Relationship Segue called view controllers, and observe how the connection and new tab appear within the canvas.
- A tab bar controller manages an array of view controllers using a relationship segue.
- Zoom in to the storyboard, select the Item bar button item in the Top App controller, use the Attributes Inspector to change the Title attribute to *Top App*, change the Image attribute to *Top App Icon*, and observe how the canvas reflects the change.
- Repeat the same procedure to add a fourth view controller to the storyboard, using *Top Movie* and *Top Movie Icon* for the tab bar button attributes.
- Run the app and interact with the four tabs.

Part 4

- Each view controller will present the title, artist, and image of a top song, album, app or movie. We have the possibility of implementing individual TopSong, TopAlbum, TopApp and TopMovie controller classes. Each controller definition would then perform the same work but with different data.
- Using Interface Builder, drag a Text Label onto the Top Song view, change the contents of the label to *Title*, and use the Attributes Inspector to customize the text alignment and font. Place the constraints to display the label where you want it to sit.
- Drag a second text label onto the Top Song view, change the contents of the label to *Artist*, and customize the alignment and font. Place the constraints to display the label where you want it to sit.
- Drag an Image View onto the Top Song view, position the image centered, below the Artist label. Use the Size Inspector to set the height and width of the image view to 200x200. Add constraints to set the height and width of the image view to 200x200 and place it under below the Artist label.
- Run the app and observe the labels appear. The image view is empty because no image has been assigned to it for display.
- Select the three interface elements by command-clicking each of them, copy and paste the three elements to the Top Album view, and center the group of elements at the top of the view. Constraints between the elements are preserved, but the constraints relative to the containing view must be recreated.
- Run the app, interact with the Top Song and Top Album tabs, and observe the labels appear.
- Duplicate the copying and pasting of the labels and image view to the Top App and Top Movie scenes, and establish the appropriate constraints.
- Add a new class to the project called `TopMediaController` that extends `UIViewController`.
- Using Interface Builder, select each view controller, and use the Identity Inspector to set the Class attribute to `TopMediaController`. One controller class will encapsulate the retrieval of RSS data, with four separate instances of the controller that will update their respective views.
- Within the `TopMediaController` implementation declare three outlet properties for the user interface components.

```
1 class TopMediaController: UIViewController {
2
3     @IBOutlet weak var titleLabel: UILabel!
4     @IBOutlet weak var artistLabel: UILabel!
```

```
5       @IBOutlet weak var imageView: UIImageView!
6       ...
```

- The properties use the weak declaration modifier because the parent view, and not the controller itself, "owns" the labels and image view; and the properties are implicitly unwrapped optional types, because the controller does not assign values to the properties during initialization.
- Using Interface Builder, expand the contents of each scene, and Control drag connections from each controller to their respective interface components. For example, Control-drag a connection from the Top Song controller to its Title label, and select the titleLabel outlet; repeat the action with each controller and its interface components.
- In the `TopMediaController` implementation, update `viewDidLoad` to modify the titleLabel text.

```
1 override func viewDidLoad() {
2     super viewDidLoad()
3     titleLabel.text = "Media Title"
4 }
```

- Run the app, interact with the tabs, and observe how each Title label appears with the contents Media Title.

Part 5

- Despite being able to specify the different label titles in Interface Builder, `TopMediaController` will eventually need to handle the display of different data for each of the four scenes. For now, `TopMediaController`'s implementation uses a hard-coded string to set the `titleLabel` text property with code.
- Add a new property to the `TopMediaController` class.

```
1 var titleText: String?
```

- The property declaration uses `var` and an optional type (`?`), because the controller class will not assign the property an initial value during initialization.
- Using Interface Builder, select the Top Song view controller and use the Identity Inspector to add a new *user defined runtime attribute* with the Key Path `titleText`, the Type `String`, and the Value `Song Title`.

- Repeat adding a new user defined runtime attribute for each respective controller, using an appropriate value for each (e.g., `Album Title`, `App Title`, `Movie Title`).
- Update the implementation of the controller `viewDidLoad` method to use the `titleText` property for `titleLabel`.
- Run the app, interact with each tab, and observe how each title label is different.
- The *user defined runtime attributes* within Interface Builder are automatically assigned to controller properties when the controllers are instantiated.
- Declaring the `titleText` property as an implicitly unwrapped optional can remove the need to force-unwrap the property before use. Modify the `titleText` property declaration and the updating of the `titleLabel` in `viewDidLoad`.

```
1 var titleText: String!
2     ...
3     titleLabel.text = titleText
4 }
```

- Run the app, interact with each tab, and observe that the functionality remains the same.
- In addition to user defined runtime attributes, Interface Builder provides a means to set controller properties through a customizable GUI in the Attributes Inspector. Add the `@IBInspectable` attribute to the `titleText` property declaration.

```
1 @IBInspectable var titleText: String!
```

- Using Interface Builder, select each view controller and use the Identity Inspector to delete each of the `titleText` user defined runtime attributes. Select each view controller, and use the Attributes Inspector to set each Title Text attribute to the appropriate value (e.g., Song Title, Album Title, App Title, Movie Title).
- The `IBInspectable` attribute informs Interface Builder to present a user interface for setting a value for the property. The Attributes Inspector uses the name and data type of the property to display the interface (e.g., `titleText` becomes Title Text).
- Run the app, interact with each tab, and observe how each title label is different.

Part 6

- Using a web browser, explore the Apple RSS feeds page: `http://www.apple.com/rss`. Click on the Top 10 Songs link, and observe the xml output. Using the web browser, modify the url in the address bar, replacing `limit=10` with `limit=1` and replacing `/xml` with `/json`, and observe the output displayed within the browser.

- XML and JSON are simple structured data formats. The RSS data embedded with the project can also be obtained from an iOS app by making a similar http request, and traversing the data structure.
- Update the `TopMediaController` `viewDidLoad` implementation with an extraction of the JSON data.

```
override func viewDidLoad() {
  super.viewDidLoad()

  let feedURL = "http://ax.itunes.apple.com/WebObjects/MZStoreServices.woa/ws/RSS/topsongs/limit=1/json"
  let request = URLRequest(url: URL(string: feedURL)!)

  let session = URLSession.shared.dataTask(with: request, completionHandler: { (data, response, error) in
    if let jsonData = data,
       let feed = (try? JSONSerialization.jsonObject(with: jsonData, options: .mutableContainers)) as? NSDictionary,
       let title = feed.value(forKeyPath: "feed.entry.im:name.label") as? String,
       let artist = feed.value(forKeyPath: "feed.entry.im:artist.label") as? String {
      self.titleLabel.text = title
      self.artistLabel.text = artist
    }
  })
  session.resume()
}
```

- Run the app, observe how the default label text appears briefly, and how the song title and artist names then appear.
- HTTP requests for RSS data can be represented with an `URLRequest` object, and how the `URL` argument uses forced unwrapping.
- The default labels in the view appear while the request for RSS data is sent asynchronously, and how the /textttcompletionHandler: argument specifies a closure that is invoked once the data is obtained from the server. Once the RSS data is retrieved, the closure casts the data to an /textttNSDictionary, and uses multiple optional bindings to navigate the structured RSS data to obtain the specific pieces of data used by the app.
- Change the implementation of `viewDidLoad` to simulate an arbitrarily long song title.
- Run the app, and observe that the song title does not fit within the bounds of the screen.

- Using Interface Builder, select the Title label within the Top Song scene, and use the Attributes Inspector to set the Autoshrink attribute to a Minimum Font Size of 10.
- Labels must have width constraints in order to infer when text content should shrink. Using Interface Builder, select the Title label within the Top Song scene and drag its left and right edges to the margin guides within the containing view. Add leading and trailing edge constraints to the Title label by Control-dragging both leftward and rightward from the label to the containing view.
- Repeat the modification of each Title label Autoshrink attribute and the addition of constraints within each scene.
- Run the app, and notice that the song title text size appears smaller, to accommodate the longer song title.
- Run the app again and observe the default Title label text appearing, and discuss how the data is still being retrieved before the label text is updated by the controller.
- Labels are updated with data once the `completionHandler:` closure is invoked.
- Using Interface Builder, select each text label, use the Attributes Inspector to ensure the Hidden attribute is checked, and observe how the labels appear lighter within the canvas. Update the implementation of the `TopMediaController` `viewDidLoad` method to enable the display of each label once the data is obtained.

```
1 ...
2 self.titleLabel.text = title
3 self.titleLabel.hidden = false
4 self.artistLabel.text = artist
5 self.artistLabel.hidden = false
6 ...
```

- Run the app, observe the song label text appear on the Top Song tab. Interact with the other tabs, and notice how each view also displays the same top song data.

Part 7

- The songs, albums, apps and movies RSS feeds all have a similar data structure, but each controller instance will need to use a different RSS feed URL.
- Within the `TopMediaController` class, change the `titleText` property to `feedURL`.
- Using Interface Builder, select a view controller, open the Identity Inspector and delete the old `titleText` user defined runtime attribute. Repeat the attribute removal for each view controller.

- Using Interface Builder, select the Top Song view controller, open the Attributes Inspector, and change the Feed URL attribute value to `http://ax.itunes.apple.com/WebObjects/MZStoreServices.woa/ws/RSS/topsongs/limit=1/json`.
- Update the inspectable Feed URL attribute for the remaining view controllers, using the appropriate url provided in the `feedurls.txt` file (in the `.zip` you downloaded.
- Update the implementation of viewDidLoad to use the feedURL property.

```
1 func viewDidLoad() {
2     super.viewDidLoad()
3     let request = URLRequest(url: URL(string: feedURL)!)
4     ...
```

- Each controller instance will rely on the particular value of its `feedURL` property to retrieve the RSS data.
- Run the app, interact with each tab, and observe how each tab displays the respective top song, album, app and movie data.

Part 8

- Examine the RSS feed data, drawing attention to the `im:image` key that contains an array of URLs for image files.
- Open an image URL in a web browser, and observe the image that appears. An image URL might be used to create another `NSURLSession` data task that retrieves the actual image data.
- Update the controller `viewDidLoad` method to retrieve the image URL from the RSS data, create an `URL` object, and pass the `URL` to a controller method that will asynchronously retrieve the image.

```
1 ...
2 let artist = feed.value(forKeyPath: "feed.entry.im:artist.↩
     label") as? String,
3 let imageURLs = feed.value(forKeyPath: "feed.entry.im:image")↩
     as? [NSDictionary] {
4   if let imageURL = imageURLs.last,
5      let imageURLString = imageURL.value(forKeyPath: "label")↩
     as? String {
6       self.loadImage(from: URL(string:imageURLString)!)
7   }
8 ...
```

- Implement the `loadImageFromURL:` method.

```
func loadImage(from URL: URL) {
  let request = URLRequest(url: URL)
  let session = URLSession.shared.dataTask(with: request, ↩
   completionHandler: { (data, response, error) in
    if let imageData = data {
      self.imageView.image = UIImage(data: imageData)
    }
  })
  session.resume()
}
```

- Run the app, interact with each tab, and observe the title, artist and image appear in each view.