Advanced Mobile Application Development
Week 7: Split Views

Having a single table view take up the entire screen on an iPad in landscape wastes space. Instead when the iPad is in landscape mode, the navigation column stays at a fixed position on the left, with the content of the selected item displayed on the right in what's known as a split-view, and applications built this way are called master-detail applications.

Split Views
A split view manages the presentation of two side-by-side panes of content in a master-detail relationship, with persistent content in the primary pane and related information in the secondary pane.
https://developer.apple.com/ios/human-interface-guidelines/ui-views/split-views/

- Split view controllers should be the root view controller for an app and can't be pushed onto a navigation stack.
- Primary pane (master): persistent information
- Secondary pane (detail): related information to the selected item in the primary pane
- The panes in a split view can contain any other type of controller.
- Before iOS 8 split view controllers were only available on the iPad
- **UISplitViewController** class https://developer.apple.com/reference/uikit/uisplitviewcontroller
  - The **viewControllers** property is an array that stores the two controllers.
- The **UISplitViewControllerDelegate** protocol manages the presentation of the child view controllers. https://developer.apple.com/reference/uikit/uisplitviewcontrollerdelegate

Display Mode
By default, a split view devotes a third of the screen(320 pixels) to the primary pane and two-thirds to the secondary pane and appears only in landscape orientation
- The width of the split in side by side is configurable
- Avoid creating a secondary pane that's narrower than the primary pane.

Split view controllers use the size class to decide how to arrange its child view controllers
- Horizontal regular: tries to display both panes side by side
  - iPad and iPhone 6 and 6s plus
- Horizontal compact: only the secondary pane is displayed. A navigation button reveals and hides the primary pane. The primary pane is layered on top of the secondary pane when visible.
  - All other iPhones

This is controlled by the displayMode property of type UISplitViewControllerDisplayMode(enum)
https://developer.apple.com/reference/uikit/uisplitviewcontrollerdisplaymode
- The display mode will be set automatically or you can change the preferredDisplayMode
- The split view controller will try to respect the display mode you specify but may not be able to accommodate that mode because of space constraints.

Modes:
- automatic
  - chooses the best display mode
- allVisible
  - both view controllers are displayed on the screen, primary on the left typically narrower than the secondary on the right.
- primaryHidden
  - the secondary view controller is displayed, the primary view controller is hidden
- primaryOverlay

- the secondary view controller is onscreen and the primary view controller is layered on top of it

UISplitViewController has a displayModeButtonItem property that is a preconfigured bar button item that changes the display mode

- When tapped it tells the split view controller to change its current display mode to the value returned by the delegate's targetDisplayModeForAction(in:) method which returns a display mode. This method is also called during rotation. (slide) Implement this method if you want to customize what display mode is returned, otherwise it will be automatic.

Popovers

A popover is a temporary view that appears above other content onscreen when you tap a control or in an area such as the bar button item in a split view. (slide)
https://developer.apple.com/ios/human-interface-guidelines/ui-views/popovers/

- Popovers can contain any view, but in a split view horizontally regular size class(portrait) it will contain the hidden left pane.
- The share functionality we just implemented uses a popover
- Use a popover to show options or information related to the content onscreen.
- When a popover is visible, interactions with other views are normally disabled until the popover is dismissed.
- A popover should be dismissed when it's no longer needed(user makes a choice, taps outside the bounds of the popover, or taps a Done button)
    - You can also dismiss the popover programatically
- UIPopoverPresentationController class (added in iOS8)
  https://developer.apple.com/reference/uikit/uipopoverpresentationcontroller
    - In nearly all cases, you use this class as-is and do not create instances of it directly.
    - UIKit creates an instance of this class automatically when you present a view controller using the UIModalPresentationPopover style.
    - Can also use the UIModalPresentationPopover style to configure a uiviewcontroller as a popover in a horizontally regular environment
    - Pre iOS8 the class was UIPopoverController
- UIPopoverPresentationControllerDelegate (added in iOS8)
  https://developer.apple.com/reference/uikit/uipopoverpresentationcontrollerdelegate
    - Lets you customize the behavior of popovers

Multitasking

Slide-over lets users overlay an app to briefly interact with it without leaving the app they're currently using. (slide)
Split-view lets users display and use two apps side-by-side

- Users can view, resize, and interact with both apps

Requirements

- iPad Air 2 or later (iPad Pro, iPad Air 2, iPad Air, iPad mini 4, iPad mini 3, iPad mini 2 (except Split View))
- Support all orientations
- Universal and compiled in iOS 9.x or later
- Uses a launch storyboard
- iPad apps created with the iOS 9 or later SDK automatically have these capabilities

If for some reason you want to opt out of allowing your app from being presented in slide over or split view configurations you need to set the UIRequiresFullScreen key to YES in your targets plist file (or check the Requires full screen box in the General tab of the target settings)

HarryPotter
File | New Project
Master-Detail Application template
HarryPotter
Device: universal

Initial Setup
Run the app in the simulator to see all that has been done already. Make sure you switch between landscape and portrait mode so you can see the split view.

Look at the Storyboard to see everything that was created.

- A split view controller as the root view controller
- A navigation controller and table view controller for the master view controller (left side)
- A navigation controller and detail view controller (right side)
- Relationship segues from the split view controller to the master and detail view controllers
- A showDetail segue from the cells in the master view controller to the detail view controller
    - A showDetail(Replace) segue is used in a splitViewController to replace the detail view with content based on a selection in the master view.
    - Different from a show(push) segue as the master is sometimes still visible (iPad landscape) and there's no ability to navigate back
- The datasource and delegate have also been set for the table view

In AppDelegate.swift a lot of work has been done for us

```
    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool
```

- An instance of the split view controller class is created and made the root view controller of the window
- Assigns the last object in the splitviewcontrollers array of controllers which is the navigation controller of our detail view to a navigation controller
- Assigns the displayModeButtonItem of the split view controller to the navigation bar of the detail view controller
- assigns the split view's delegate (for a split view controller you can't do this in the storyboard)

```
    func splitViewController(_ splitViewController:
UISplitViewController, collapseSecondary
secondaryViewController:UIViewController, onto
primaryViewController:UIViewController) -> Bool
```

- this method performs the tasks related to the transition to a collapsed interface
- adjusts the primary view controller and incorporate the secondary view controller into the collapsed interface
- a compact horizontal size class (iPhone) will collapse the split view
- When the split view collapses what happens to the second view controller is up to the second view controller (return false)

- Removes the second view controller and have the first view controller as the only child of the split view controller return true. This case is so the master view controller is shown on the iPhone when the app launches
- return values
    - "false" tells the split view controller to use its default behavior to try and incorporate the secondary view controller into the collapsed interface
    - "true" tells the split view controller not to apply the default behavior, that you do not want the split view controller to do anything with the secondary view controller

This template created a MasterViewController class and a DetailViewController class. These represent, respectively, the views that will appear on the left and right sides of the split view.

MasterViewController.swift
MasterViewController is a subclass of UITableViewController and defines an instance for the detail view controller.

viewDidLoad()
adding the edit and + buttons to the navigation bar.
Setting the detailviewcontroller

viewWillAppear()
The value of `clearsSelectionOnViewWillAppear` is based on whether or not the split view is collapsed.
By default, UITableViewController is set up to deselect all rows each time it's displayed. That may be OK in an iPhone app, where each table view is usually displayed on its own; however, in an iPad app featuring a split view, you probably don't want that selection to disappear.

prepare(for segue:)
Handles the segue and has all the methods for the table view.

DetailViewController.swift
DetailViewController is a subclass of UIViewController
Defines the outlet detailDescriptionLabel for the label in the storyboard.
Creates a variable called detailItem where the view controller stores its reference to the object that the user selected in the master view controller. The code in the didSet block is called after its value has been changed and calls configureView(), another method that's generated for us. configureView() to update the label with the detail item.

Wow, a lot of the setup was done for us. Now let's get going on our app.

Data
Drag in the harrypotter.plist file we're using and remember to check Copy items if needed.
Look at the plist to understand what's in it and what the key/value pairs look like.

In MasterViewController we need an array that will hold our list of characters. Replace the one in the template. This matches the format of our plist.

```
var characters = [[String : String]]()
```

Then we need to load in our plist data using the method we've been using.

```
    func getDataFile() -> String? {
        //use a Bundle object of the directory for our application to
retrieve the pathname of artistalbums.plist
        guard let pathString = Bundle.main.path(forResource: "harrypotter",
ofType: "plist") else {
            return nil
        }
        return pathString
    }
```

In viewDidLoad remove(or comment out) the code that's in there to allow edits since we won't be doing that.

```
        guard let path = getDataFile() else{
            print("Error loading file")
            return
        }

        //load the data of the plist file into the dictionary
        let alldata = NSDictionary(contentsOfFile: path) as! [String:
[[String : String]]]
        //print(alldata)
        if alldata.isEmpty != true {
            characters = Array(alldata["characters"]!)
        }
```

We also need to remove some table view delegate and data source methods.
Delete/comment out these whole methods:

```
func insertNewObject(_ sender: AnyObject)

func tableView(tableView: UITableView, commitEditingStyle editingStyle:
UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath)
```

Now let's update the other table view data source methods (stubs)
We have 1 section so `numberOfSections(in tableView)` is fine

```
 func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return characters.count
    }
```

Update `func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {`

```
     return false
    }
```

And now configure the appearance of the cell

```
    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "Cell",
for: indexPath)
        let character = characters[indexPath.row]
```

```
        cell.textLabel!.text = character["name"]! as String
        return cell
    }
```

Lastly we have to update the prepare(for segue:) to pass the URL to the detail view controller.

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "showDetail" {
            if let indexPath = self.tableView.indexPathForSelectedRow {
                let character = characters[indexPath.row]
                let url = character["url"]! as String
                let name = character["name"]! as String
                let controller = (segue.destination as!
UINavigationController).topViewController as! DetailViewController
                controller.detailItem = url as AnyObject?
                controller.title = name
                controller.navigationItem.leftBarButtonItem =
self.splitViewController?.displayModeButtonItem
                controller.navigationItem.leftItemsSupplementBackButton =
true
            }
        }
    }
```

When you run it tap the Master button in the upper-left corner to bring up a popover with a list of characters. Tap a character's name to display their Wikipedia page URL in the detail view.

Rotate to landscape and you'll see the characters in the table view in the master view.

Detail View

In the Master view controller change its title to "Characters" instead of it saying Master.
Now let's get the detail view working.
Remove the title 'Detail' so that doesn't show when the app first starts.
Go into the storyboard and move the existing label to the top just under the nav bar and update its constraints.(replace center Y constraint with top space to top layout 0)
Change it to say "Select a Harry Potter character". We will use this label to show the URL but before the user selects anything it will provide instructions.
Look in the connection inspector to see that it's already set up as an outlet called detailDescriptionLabel.
Then add a web view that fills up the rest of the view and make an outlet connection called webView.
In the attributes inspector for Scaling check Scales Page to Fit.
Also connect the delegate for the webView to the DetailViewController.
Add constraints so leading and trailing are to the edges and top space is to the label(16 to top layout).

In DetailViewController.swift add a method to load the web page.

```
    func loadWebPage(_ urlString: String){
        //the urlString should be a propery formed url
        //creates a NSURL object
        let url = URL(string: urlString)
        //create a NSURLRequest object
        let request = URLRequest(url: url!)
        //load the NSURLRequest object in our web view
```

```
        webView.loadRequest(request)
    }
```

Call loadWebPage() from configureView

```
    func configureView() {
        // Update the user interface for the detail item.
        if let detail = self.detailItem {
            if let label = self.detailDescriptionLabel {
                label.text = detail.description
                loadWebPage(detail.description)
            }
        }
    }
```

You should now be able to select a character and see their Wikipedia page in the detail.
To change the title of the back button for the popover in portrait mode change the Title for the master view controller in the Storyboard in its navigation controller.

Activity Indicator

Add an activity indicator in the middle and connect it as an outlet called webSpinner
Make it gray and check Hide When Stopped and Animating(otherwise it will be hidden).
For the activity indicator align horizontal and vertical centers with the web view.
Make sure in the document outline that the activity spinner is below the web view.
Go into DetailViewController .swift and add the `UIWebViewDelegate` protocol. All the methods for this protocol are optional (look at class reference).
Then add the two methods for the activity indicator

```
//UIWebViewDelegate method that is called when a web page begins to load
    func webViewDidStartLoad(_ webView: UIWebView) {
webSpinner.startAnimating()
    }

    //UIWebViewDelegate method that is called when a web page loads
successfully
    func webViewDidFinishLoad(_ webView: UIWebView) {
webSpinner.stopAnimating()
    }
```

Another useful delegate method is `webView(_:, didFailLoadWithError:)` which is called if the web page doesn't load (such as no internet connection).
We won't be implementing it in this app.

Don't forget to add app icons and a launch screen.
Note that for universal you need iPhone and iPad app icons. iPad are 76 1x and 2x, and 83.5 2x for iPad Pro.