Advanced Mobile Application Development
Week 4: Navigation Controllers Insert, Static Cells

Add countries
Let's build on countries so we can add countries as well.
(countries add)
Go into Mainstoryboard and add a new view controller.
Add a label and a textfield in it where the user can enter a new country.

Create a new Cocoa Touch class to be its controller called AddCountryViewController, subclass of UIViewController.
Back in the storyboard make the AddCountryViewController class the controller for the new view controller.
Now let's make an outlet connection for the textfield called countryTextfield. Remember you must make this connection to AddCountryViewController.swift.
Back in the storyboard go into the Detail view and add a navigation item. Remove its title.
Add a bar button item to the navigation item and change its System Item to Add.
Go into DetailViewController.swift and in ViewDidLoad comment out
```
self.navigationItem.rightBarButtonItem=self.editButtonItem()
```

Go back into the storyboard so we can create a segue from the Detail to AddCountry view controllers.
Control click from the add button and drag to the country view controller and choose a present modally segue.
We don't want push navigation because that's designed for a drill-down interface, where you're providing more information about whatever the user selected. Adding an item is a modal operation—the user performs some action that's complete and self-contained, and then returns from that scene to the main navigation.

If you run it now the add button will bring up the country view controller, but you'll notice there's no way for it to go back. Because a modal view controller doesn't get added to the navigation stack, it doesn't get a navigation bar from the table view controller's navigation controller. However, you want to keep the navigation bar to provide the user with visual continuity. To give the country view controller a navigation bar when presented modally, embed it in its own navigation controller.
With the AddCountry view controller selected, choose Editor > Embed In > Navigation Controller.

Along with adding a navigation controller, it gave the addcountry view controller a nav bar.
Make its title Add New Country and add two bar button items. Put one on the right and change it to Done, and the other on the left and change it to Cancel.

If you run it at this point you should see the nav bar with the buttons but they don't do anything yet.
You'll see we need to fix the constraints for our addcountry view.

Let's implement the Cancel button by setting up an unwind segue to undo the modal segue back to the detail view controller.
First we have to create an unwind method in the destination view controller, DetailViewController. A method that can be unwound to must return an IBAction and take in a UIStoryboardSegue as a parameter.
```
    @IBAction func unwindSegue (segue:UIStoryboardSegue){ }
```
We will implement it later.

Now go back into the storyboard and from the cancel button in the add country scene, control click and connect to the Exit icon in the dock and choose this unwind method.
Choose this segue and give it the identifier cancelSegue.
Do the same thing for the Done button. Choose this segue and give it the identifier doneSegue.
Both will call the same method, we'll distinguish between the two different cases when we write the code to handle the segue.

If you run it now the done and cancel buttons will both take you back to the detail view controller, but your data is not saved yet.

Add Rows
Go into AddCountryViewController.swift

We are defining a string to store the addedCountry so that DetailViewController will have access to it.

```
var addedCountry = String()

    override func prepareForSegue(segue: UIStoryboardSegue, sender:
AnyObject?) {
        if segue.identifier == "doneSegue"{
            //only add a country if there is text in the textfield
            if ((countryTextfield.text?.isEmpty) == false){
            addedCountry=countryTextfield.text!
            }
        }
    }
```

If the user leaves the textfield empty and hits Done an empty row will be added. You can tell it adds an empty row because if you swipe that row you get the delete option (which you don't get on a row that doesn't exist). We add the .isEmpty test to avoid adding empty rows.

In DetailViewController we implement the unwind method

```
    @IBAction func unwindSegue(segue:UIStoryboardSegue){
        if segue.identifier=="doneSegue"{
            let source = segue.sourceViewController as!
AddCountryViewController
            //only add a country if there is text in the textfield
            if ((source.addedCountry.isEmpty) == false){
                countries.append(source.addedCountry)
                tableView.reloadData()
                let chosenContinent =
continentListDetail.continents[selectedContinent]

continentListDetail.continentData[chosenContinent]?.append(source.addedCount
ry)
            }
        }
    }
```

Run your app and you should be able to add new countries. Try to add an empty row. When you swipe you don't get the Delete icon which means it wasn't added.

Continent Info
Now let's add another table view this time using static cells.

In the continents scene(master) change the accessory to a detail disclosure in the table view cell.
This is so selecting a row will still drill down to the countries, but tapping on the accessory will bring us to a new table view with static cells.

Drag a table view controller onto the storyboard.
Create a new class(Cocoa Touch) for it called ContinentInfoViewController. Make sure you make its superclass UITableViewController.
Back in the storyboard set the class for the new scene to be ContinentInfoViewController.

Create a segue from the master continents cell to the new country info view controller and choose Accessory Action show segue. Give it the idenitifer continentsegue.
Now when you run it selecting the detail disclosure indicator brings you to the new detail view.

Static cells
Now let's get our static cells set up in the continent info scene.
Select the table view in the continent info scene and in the Attributes Inspector change Content to Static Cells and Style to Grouped.
Select the Table View Section(document hierarchy) and change Rows to 2 and header to Continent Info.

Select the first cell and use the attributes inspector to set its Style to Right Detail. Double-click to select the text of the label on the left and change it to Continent. Repeat the same steps for the second cell, changing its text to Number of countries.

Now we're going to create outlets for the detail labels called continentName and countryNumber. Make sure you're making the connections to ContinentInfoViewController.swift and that you are making them from the detail labels(use the document hierarchy to check).

Go into ContinentInfoViewController and add two variables

```
    var name = String()
    var number = String()
```

Now delete the 2 dataSource methods as static cells don't use them.

```
func numberOfSectionsInTableView(tableView: UITableView!) -> Int

func tableView(tableView: UITableView!, numberOfRowsInSection section: Int)
-> Int
```

Now if you run it you can see your static cells.
Now let's populate them with data.

```
    override func viewWillAppear(animated: Bool) {
        continentName.text=name
        countryNumber.text=number
    }
```

In ViewController.swift we have to update prepareForSegue to work with the detail disclosure accessory button by adding the following after the if statement:

```
        else if segue.identifier == "continentsegue"{
            let infoVC = segue.destinationViewController as!
ContinentInfoTableViewController
            let editingCell = sender as! UITableViewCell
            let indexPath = tableView.indexPathForCell(editingCell)
            infoVC.name = continentList.continents[indexPath!.row]
            let countries = continentList.continentData[infoVC.name]! as
[String]
            infoVC.number = String(countries.count)
        }
```

Data Persistence

Now let's save our data to a plist so it's persistent.
ViewController.swift
Define a constant for our data file

```
let kfilename = "data.plist"
```

Now we write a method that will return the path to a given file.

```
    func docFilePath(filename: String) -> String?{
        //locate the documents directory
        let path =
NSSearchPathForDirectoriesInDomains(NSSearchPathDirectory.DocumentDirectory,
NSSearchPathDomainMask.AllDomainsMask, true)
        let dir = path[0] as NSString //document directory
        //creates the full path to our data file
        return dir.stringByAppendingPathComponent(filename)
    }
```

Update viewDidLoad so we see if data.plist exists and if it does we use that. If it doesn't, we use our initial plist.
Then our application needs to save its data before the application is terminated or sent to the background, so we'll use the UIApplicationWillResignActiveNotification notification. This notification is posted whenever an app is no longer the one with which the user is interacting. This includes when the user quits the application and (in iOS 4 and later) when the application is pushed to the background.

```
    override func viewDidLoad() {
        super.viewDidLoad()
        let path:String?
        let filePath = docFilePath(kfilename) //path to data file

        //if the data file exists, use it
        if NSFileManager.defaultManager().fileExistsAtPath(filePath!){
            path = filePath
            print(path)
        }
        else {
            //use a NSBundle object of the directory for our application to
retrieve the pathname of our initial plist
```

```
            path = NSBundle.mainBundle().pathForResource("continents",
ofType: "plist")
            print(path)
        }

        //load the data of the plist file into the dictionary
        continentList.continentData = NSDictionary(contentsOfFile: path!)
as! [String : [String]]
        //puts all the continents in an array
        continentList.continents = Array(continentList.continentData.keys)

        //application instance
        let app = UIApplication.sharedApplication()
        //subscribe to the UIApplicationWillResignActiveNotification
notification
        NSNotificationCenter.defaultCenter().addObserver(self, selector:
"applicationWillResignActive:", name:
"UIApplicationWillResignActiveNotification", object: app)
    }
```

Now we'll create the notification method `applicationWillResignActive`

```
    //called when the UIApplicationWillResignActiveNotification notification
is posted
    //all notification methods take a single NSNotification instance as
their argument
    func applicationWillResignActive(notification: NSNotification){
        let filePath = docFilePath(kfilename)
        let data = NSMutableDictionary()
        //adds our whole dictionary to the data dictionary
        data.addEntriesFromDictionary(continentList.continentData)
        print(data)
        //write the contents of the array to our plist file
        data.writeToFile(filePath!, atomically: true)
    }
```

We need to use a NSMutableDictionary because the NS classes have the writeToFile() methods. Mutable means it can be changed.

To test the app fill in the text fields, press the home button, then exit the simulator and run it again, it should load your data. If you just exit the simulator without pressing the home button, that's the equivalent of forcibly quitting your application. In that case, you will never receive the notification that the application is terminating, and your data will not be saved.