Table View

Table views are used to display lists of information. Combined with navigation controllers, they can be used to display a hierarchy of information.

Table views are essentially scroll views with some automatic behavior, like retrieving data, figuring out which ones need to be rendered on-screen, managing the memory used by the cells, responding to user gestures like flicks, swipes, and taps, rendering animated deletes and inserts on the table, and so on. They are an essential part of the iOS user interface paradigm.

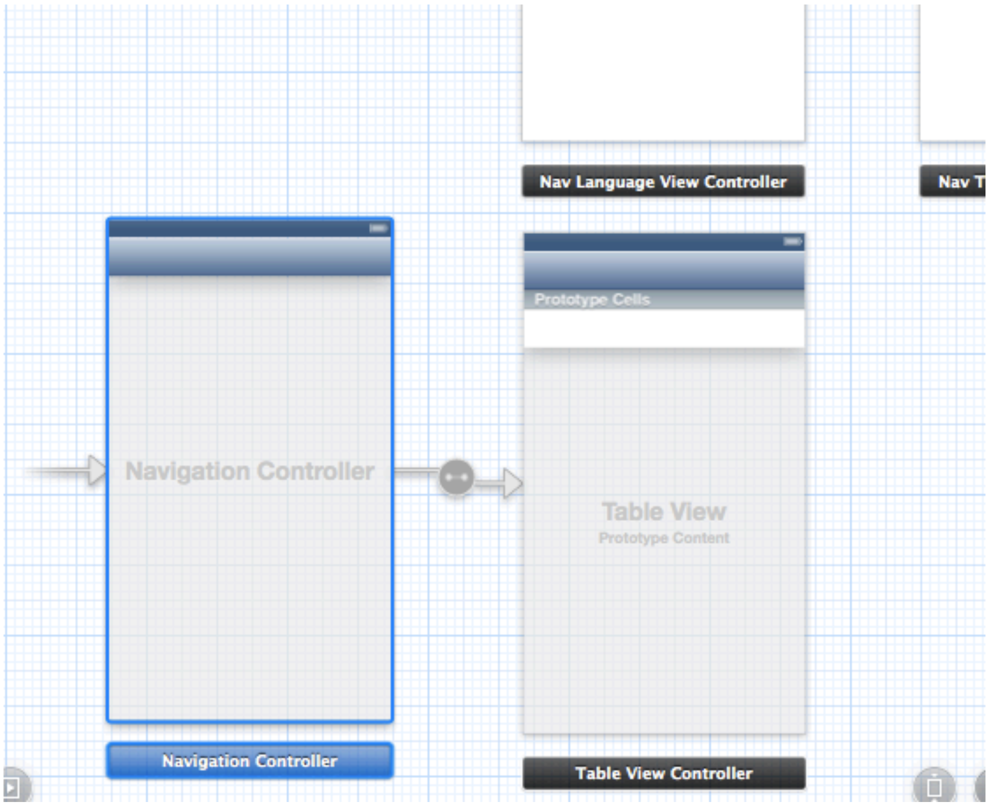
Table views can be used to display *static* or *dynamic* information.

Static Table Views

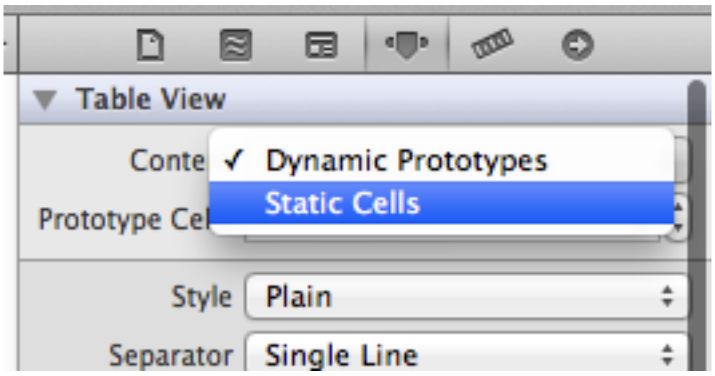
In our previous project, we created two scenes, the first scene containing buttons allowing us to select a language, which segues into a second scene that displays the translation of Hello World in the chosen language.

Let's modify this app to use static table views instead.

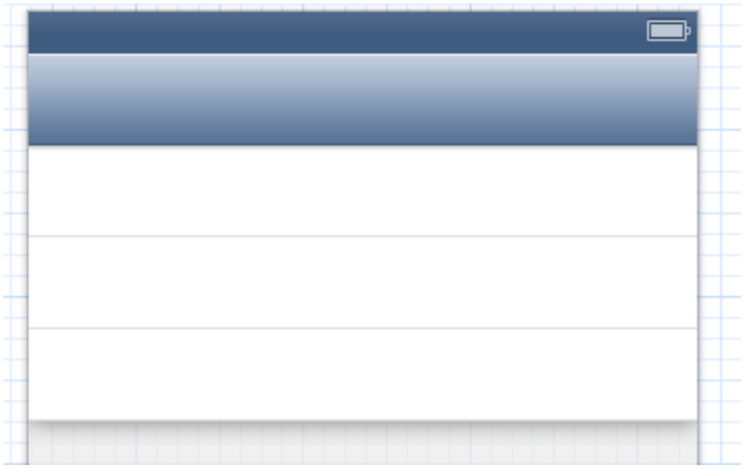
First, go to the storyboard and drag over a Table View Controller from the objects palette. You can go ahead and re-arrange the scenes so that the Navigation Controller is right next to the new table view controller. Now change the Navigation Controller's root view by Control dragging from the Navigation Controller to the Table View Controller. You should get something like this:



Next we change the table view to static by clicking on the table view and in the attributes inspector, changing the Content type from Dynamic Prototypes to Static Cells.

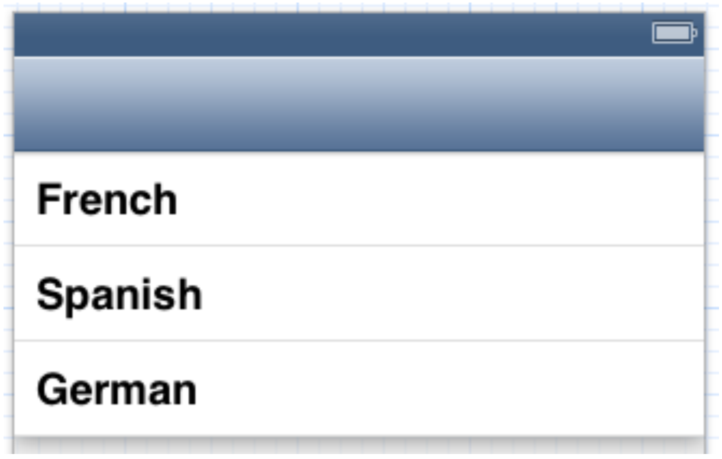


Your table view should now look something like this:



In terms of code, a Table View Controller is essentially a sub-class of UIViewController that implements the UITableViewDataSource and UITableViewDelegate protocols (which we'll discuss in more detail later). In the storyboard, the Table View Controller comes with a UITableView, and in the case of Static Cells, the TableView comes with three UITableViewCells.

This gives us three cells in which to display information, each of which is type "Custom". For now, simply select the Basic layout for each cell, and type in the languages we had earlier:



Note that at this point you can also change the number of sections (if you were doing a categorized list) by clicking on the table view and specifying the number of sections in the Attributes inspector. For example:

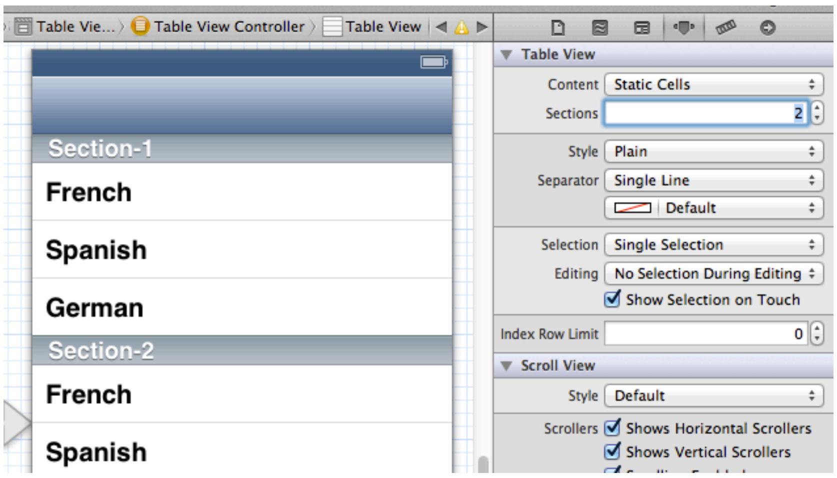
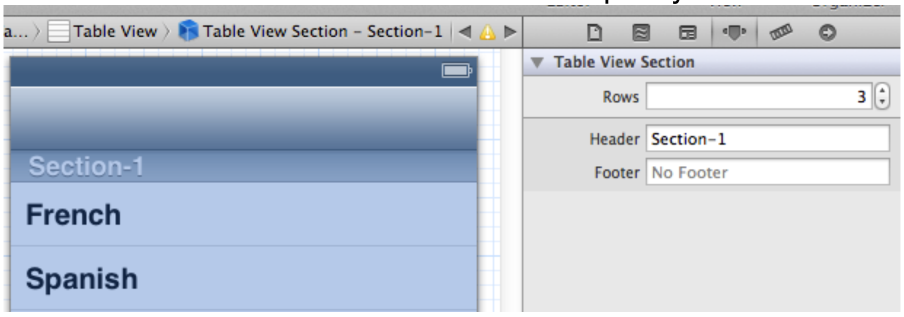


Table Vie… › Table View Controller › Table View

Section-1
French
Spanish
German
Section-2
French
Spanish

Table View
Content: Static Cells
Sections: 2
Style: Plain
Separator: Single Line
Default
Selection: Single Selection
Editing: No Selection During Editing
✓ Show Selection on Touch
Index Row Limit: 0
Scroll View
Style: Default
Scrollers: ✓ Shows Horizontal Scrollers
✓ Shows Vertical Scrollers

You can change the section attributes by clicking on the section header and changing the header text and footer text. You can also specify the number of rows here.


a… › Table View › Table View Section – Section-1

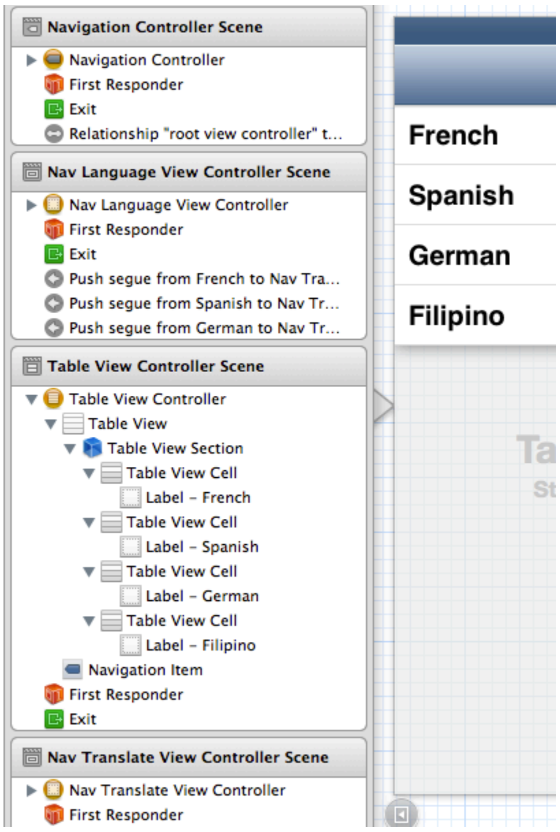
Section-1
French
Spanish

Table View Section
Rows: 3
Header: Section-1
Footer: No Footer

HINT: If you only have one section, and wanted to add rows to it, you'd have a hard time since there is no section header being displayed. In this case, you can display the outline view by clicking on the (>) in the lower left corner of the storyboard editor, and selecting the section there

You can also cut and paste your cells to add more cells with the same attributes.

Here we've added a fourth cell with the label Filipino.


Navigation Controller Scene
- Navigation Controller
- First Responder
- Exit
- Relationship "root view controller" t…

Nav Language View Controller Scene
- Nav Language View Controller
- First Responder
- Exit
- Push segue from French to Nav Tra…
- Push segue from Spanish to Nav Tr…
- Push segue from German to Nav Tr…

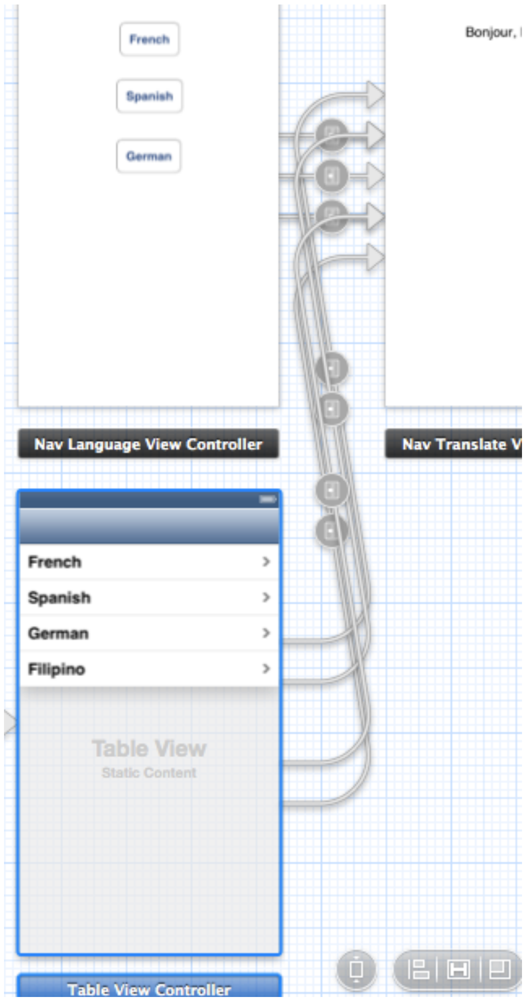
Table View Controller Scene
- Table View Controller
  - Table View
    - Table View Section
      - Table View Cell
        - Label – French
      - Table View Cell
        - Label – Spanish
      - Table View Cell
        - Label – German
      - Table View Cell
        - Label – Filipino
  - Navigation Item
- First Responder
- Exit

Nav Translate View Controller Scene
- Nav Translate View Controller
- First Responder

French
Spanish
German
Filipino

Now let's add segues!

Simplest thing to do is to create a segue from each cell into the NavTranslateViewController that we've already created previously. So here, we've created four push segues by control dragging.



We also need to give these segues identifiers. We can go ahead and use the same identifiers from before (French, Spanish, German, Filipino) since these identifiers do not need to be unique.

Click on each segue and type in the appropriate identifier into the attributes inspector just like before.

When you click on each segue, pay attention to what gets highlighted, since that's telling you where the segue is coming from.

Also notice the little > that appears as you create the segue. This is called the disclosure indicator, and its something IB (Interface Builder) puts in to indicate that clicking the cell will go somewhere. You can control this by clicking on the cell and going into attributes inspector to see what other kinds of things can be displayed there.

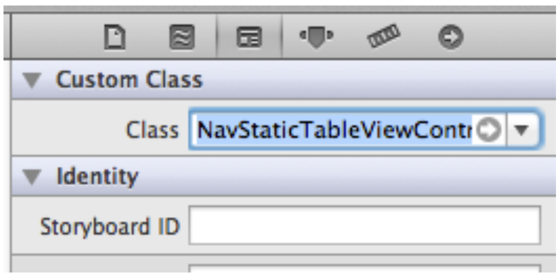
Go ahead and run your app. Notice that when you tap the cell, it goes into a blank view. That's because we haven't put in the code that will pass the data on to the next view controller.

To do that, we need to create a new subclass.

This time, however, instead of subclassing UIViewController, we will be subclassing UITableViewController.

So follow the steps as before, File->New->File..., select Objective-C class, call it NavStaticTableViewController, make sure it's a sub-class of UITableViewController, and create the files.

Go back to the storyboard and change the Custom class of our Table View Controller to NavStaticTableViewController.


▼ Custom Class

Class NavStaticTableViewContr

▼ Identity

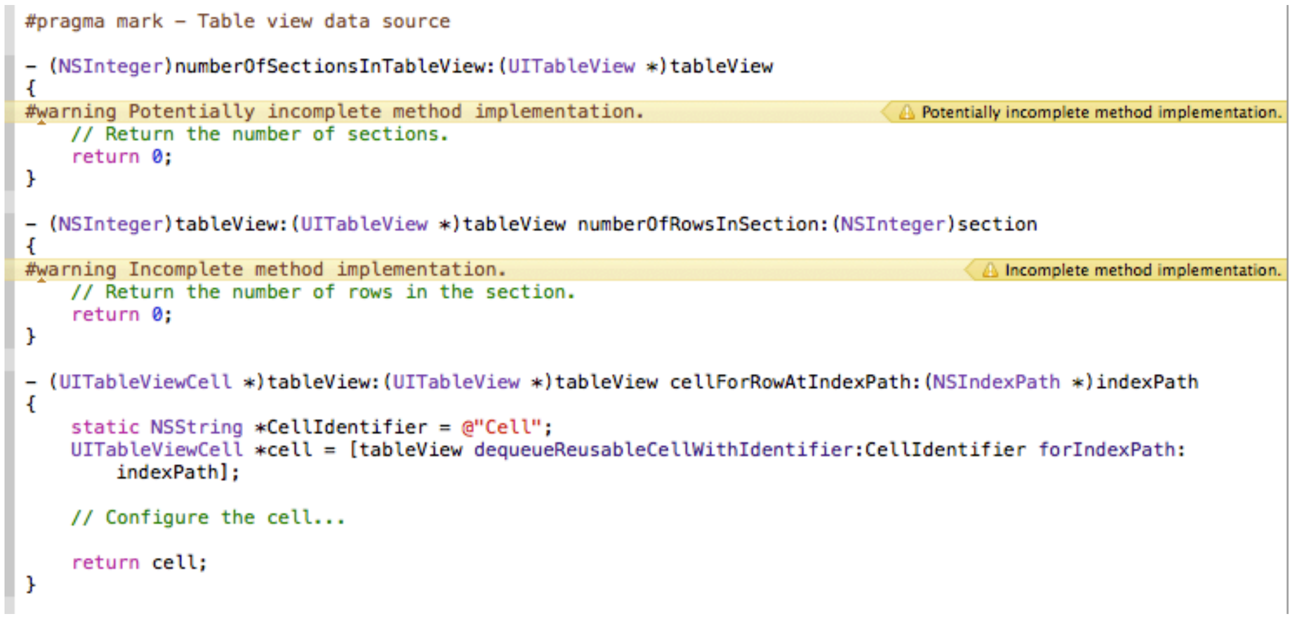
Storyboard ID

Now try running the app again.

You'll notice that instead of our table view with the languages, you'll instead get a blank table view.

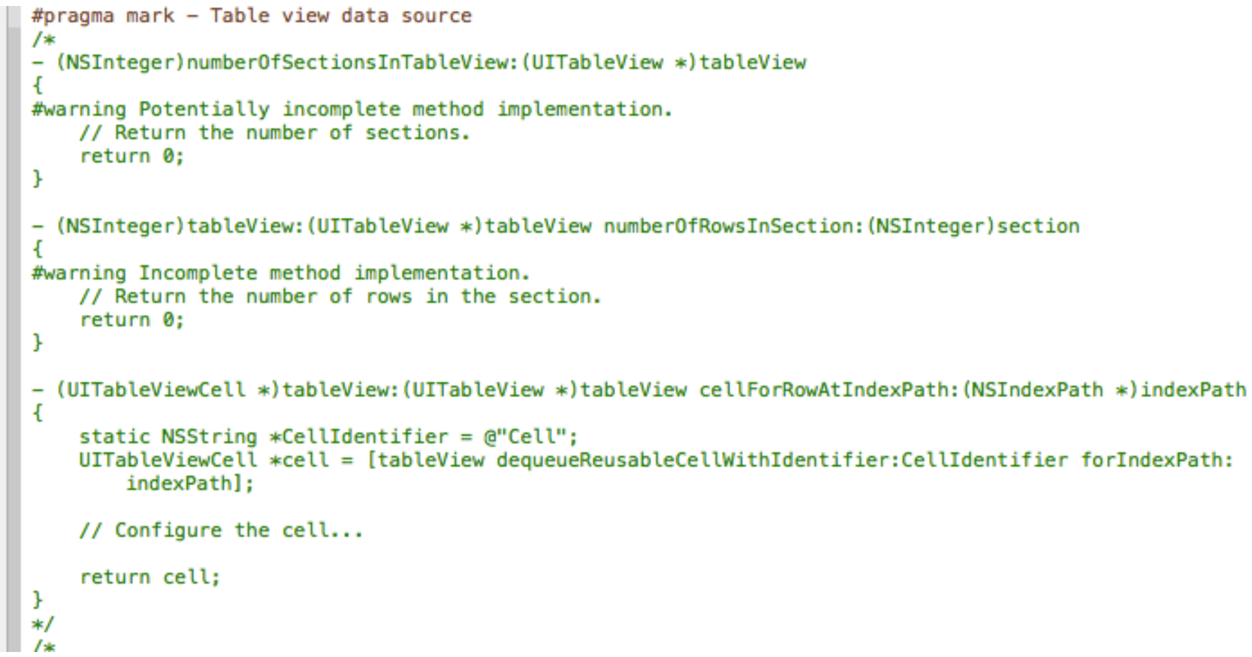
The reason for this is that the default template for UITableViewController subclass assumes a dynamic table view. In order to get it working for static table views, we need to go into the code and comment out a few things.

Look for this part and comment it out:



```
#pragma mark - Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
#warning Potentially incomplete method implementation.          ⚠ Potentially incomplete method implementation.
    // Return the number of sections.
    return 0;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
#warning Incomplete method implementation.                      ⚠ Incomplete method implementation.
    // Return the number of rows in the section.
    return 0;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:
        indexPath];

    // Configure the cell...

    return cell;
}
```

After commenting out it should look like this:



```
#pragma mark - Table view data source
/*
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
#warning Potentially incomplete method implementation.
    // Return the number of sections.
    return 0;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
#warning Incomplete method implementation.
    // Return the number of rows in the section.
    return 0;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier forIndexPath:
        indexPath];

    // Configure the cell...

    return cell;
}
*/
/*
```

Now try running your app again. It should be back where it was before.

Now the only thing left to do is to deal with the segues and pass data based on the segue's identifier. Sound familiar? Yep, we can simply copy over our code from before for prepareForSegue and add the new language Filipino.

First remember to #import the NavTranslateViewController.h

```
#import "NavStaticTableViewController.h"
#import "NavTranslateViewController.h"
```

Then here's the code for prepareForSegue that you need to paste just before the @end of the .m file:

```
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    NavTranslateViewController *vc = [segue destinationViewController];
    if ([[segue identifier] isEqualToString:@"French"]) {
        // translate to French
        vc.translatedText = @"Bonjour, Monde!";
    }
    if ([[segue identifier] isEqualToString:@"Spanish"]) {
        // translate to Spanish
        vc.translatedText = @"Hola, Mundo!";
    }
    if ([[segue identifier] isEqualToString:@"German"]) {
        // translate to German
        vc.translatedText = @"Hallo, Welt!";
    }
    if ([[segue identifier] isEqualToString:@"Filipino"]) {
        // translate to German
        vc.translatedText = @"Mabuhay, Mundo!";
    }
}
```

Let's add one last touch and add a title to our static table view controller. We can do this in storyboard or in the viewDidLoad:

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Uncomment the following line to preserve selection between
presentations.
    // self.clearsSelectionOnViewWillAppear = NO;

    // Uncomment the following line to display an Edit button in the
navigation bar for this view controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem;

    self.navigationItem.title = @"Hello World!";

}
```

Now run the app and marvel at your handiwork.

ON YOUR OWN: How about changing the title of the NavTranslateViewController to display the selected language? Hint: you might need to pass another property in containing the language.