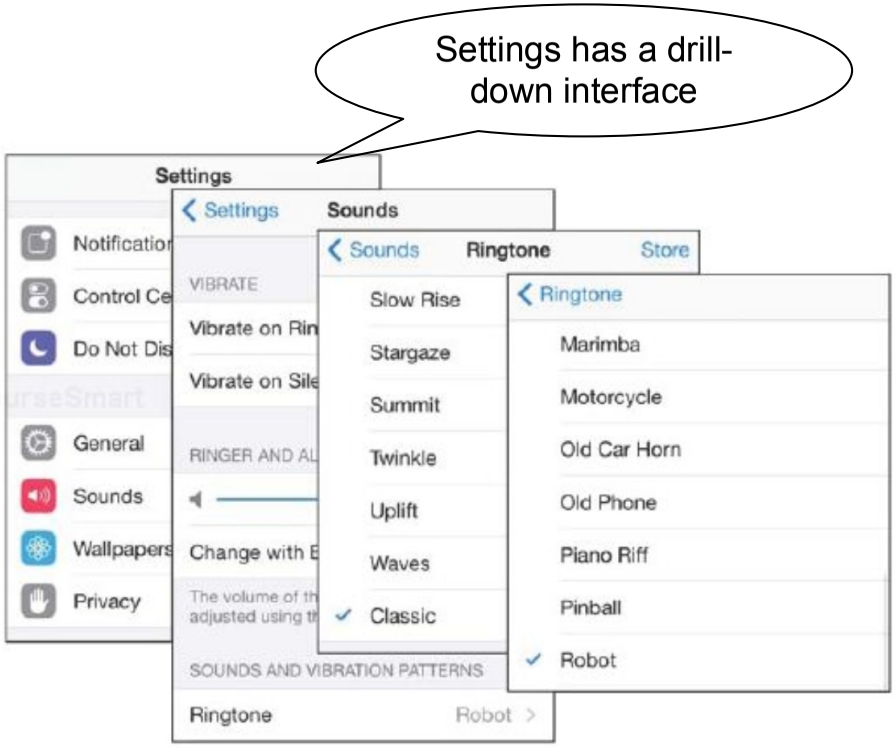
Chapter 10
UINavigationController

- UINavigationController
- Navigating with UINavigationController
- UINavigationBar

UINavigationController

- UINavigationController is used when our application presents multiple screens of information, typically in a drill-down fashion.
- We will use a UINavigationController to add a drill- down interface to Homeowner that lets the user view and edit the details of a INIItem



- In this chapter:

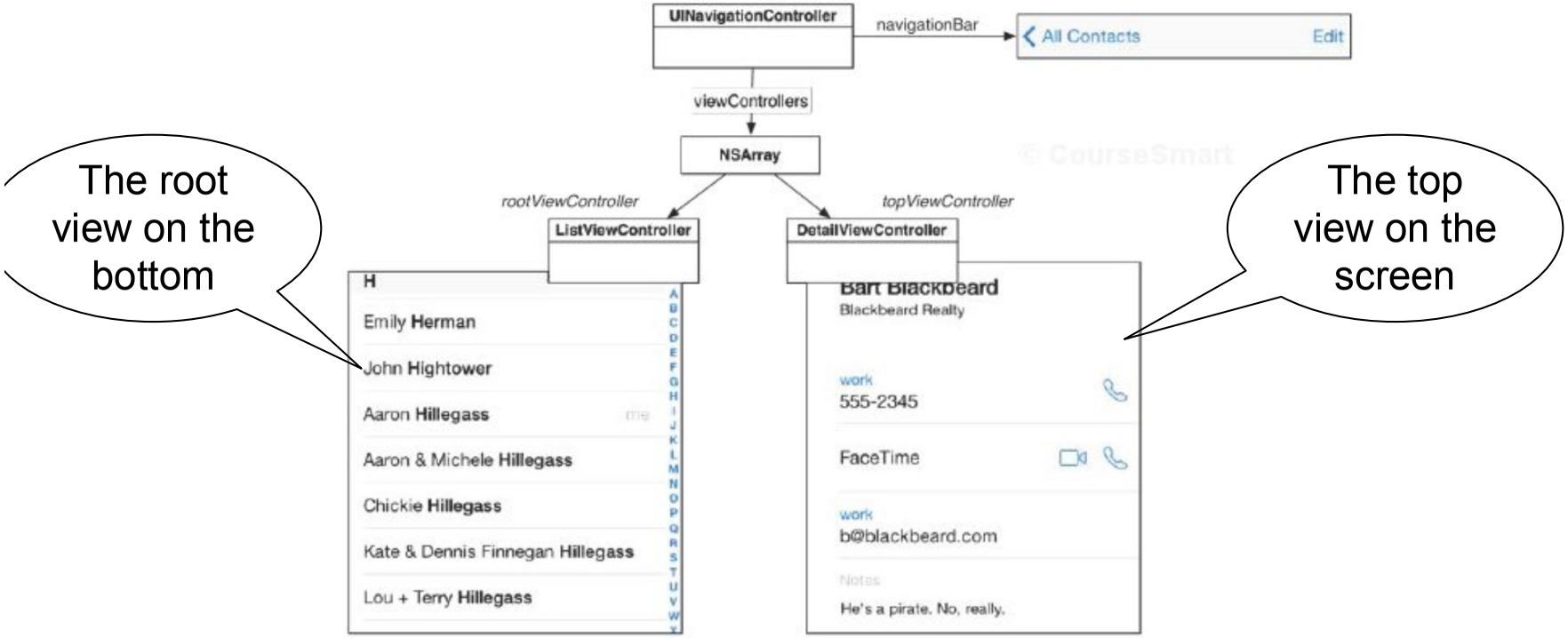
  We you will add a UINavigationController to the Homeowner application and make the INIItemsViewController the UINavigationController's rootViewController.

  We will create another subclass of UIViewController that can be pushed onto the UINavigationController's stack and will allow the user to edit the properties of the selected INIItem.

  When a user selects one of the rows, the new UIViewController's view will slide onto the screen.

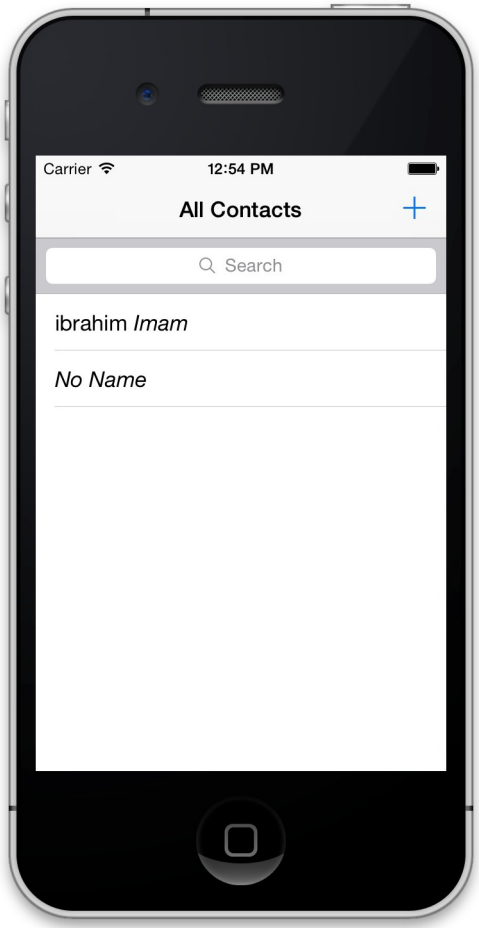
UINavigationController

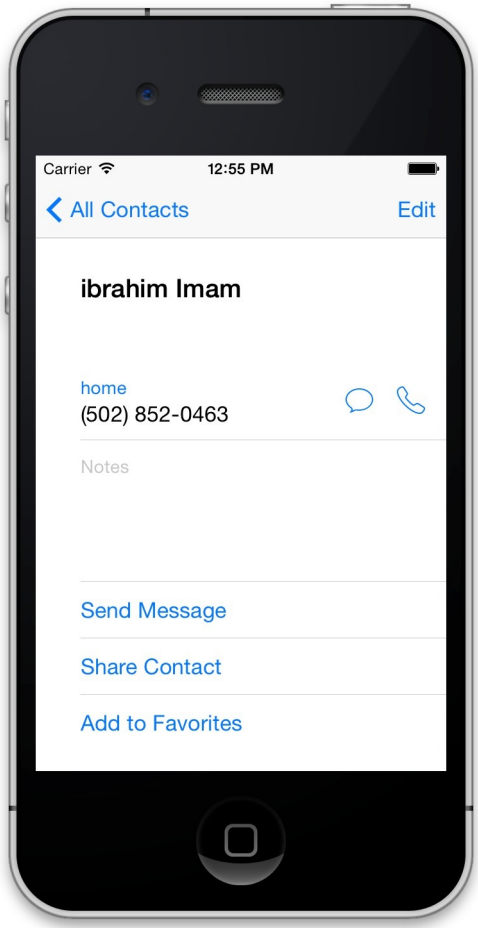
- UINavigationController maintains a stack of the screens where each screen is the view of a UIViewController.
- The stack is an array of view controllers.
- The view that is visible is that of the view controller on top of the stack
- When we initialize UINavigationController we give it one UIViewController which is the root view of the navigator and it is always on the bottom of the stack.
- More view controllers can be pushed on top of the UINavigationController's stack while the application is running. The Top of the stack is the bottom of the array



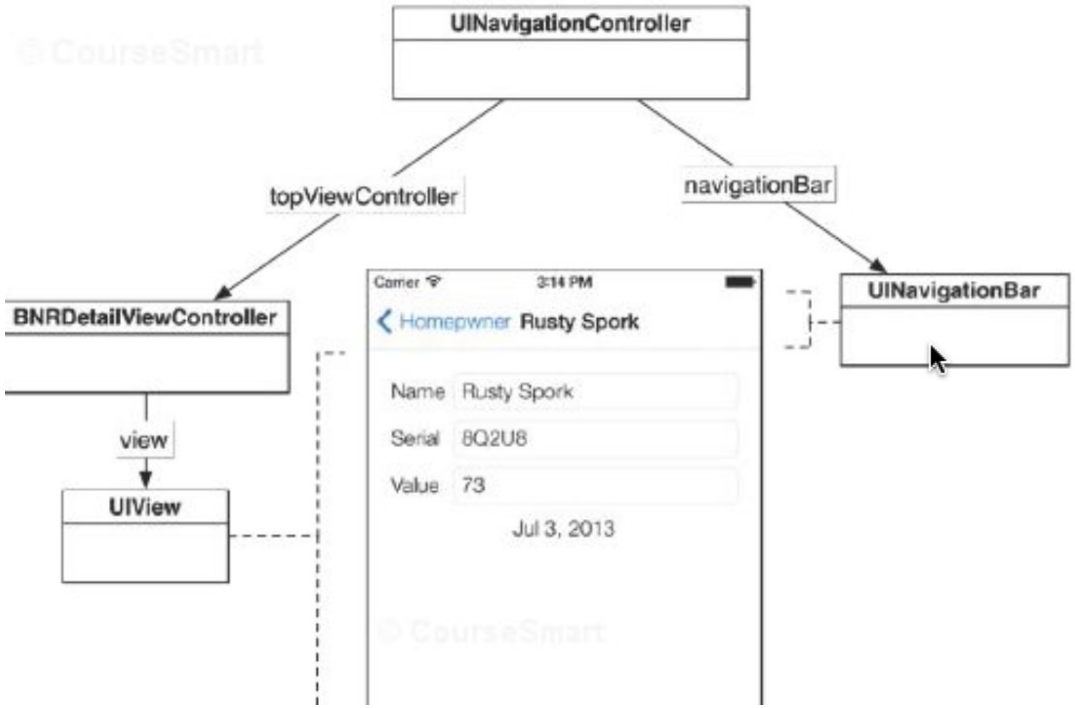
UINavigationController

- When a view is pushed on the stack it slides onto the screen from the right.
- When a view is pop of the stack it slides off the screen and the view below it slides in view from the left.



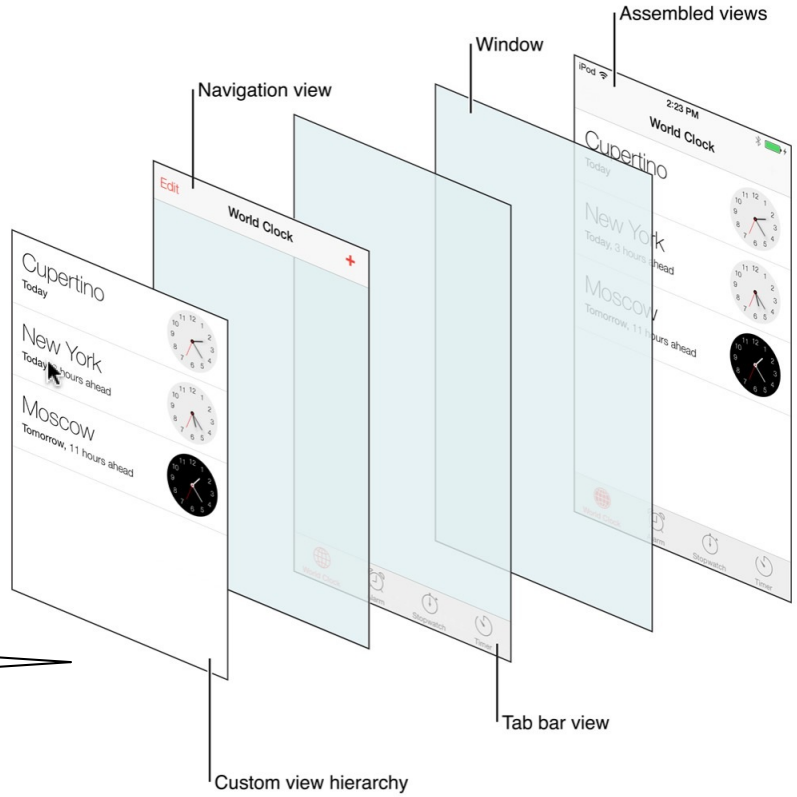


The views of a navigation controller

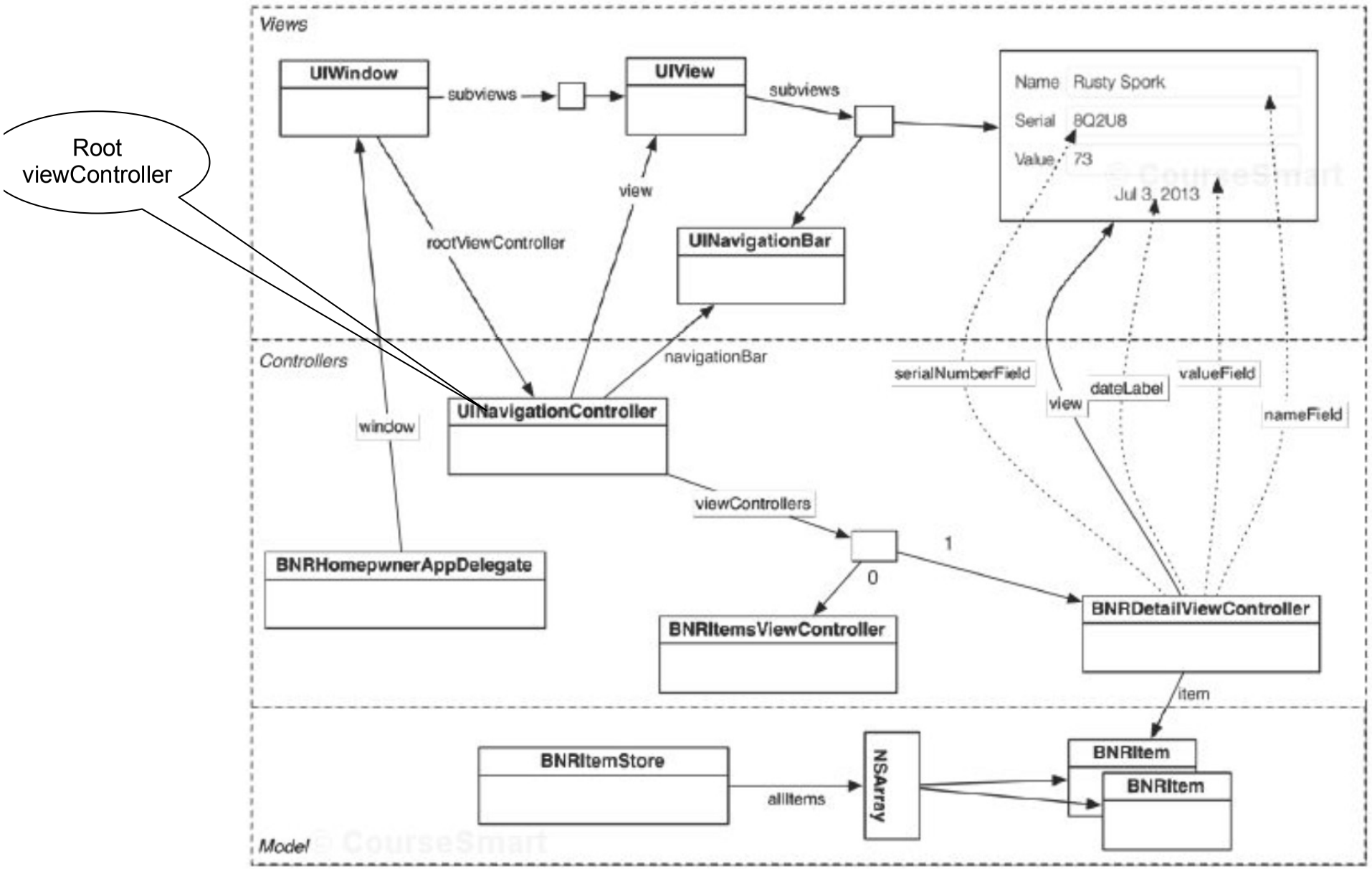


A navigation controller has:

- Navigation Bar
- Navigation View (where custom contents are placed
- Navigation toolbar



The Homeowner object diagram

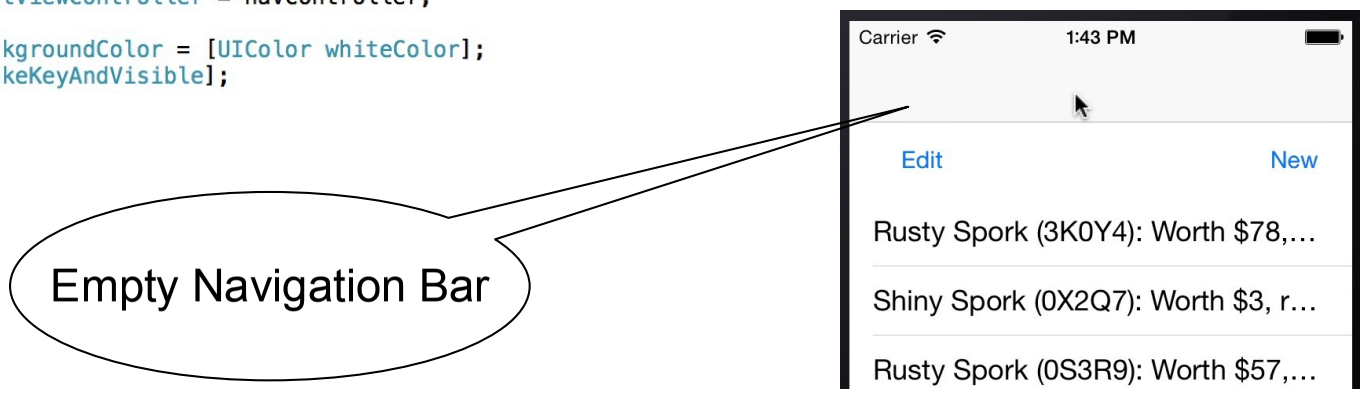


The Homeowner application

Make the following changes to application:didFinishLaunchWithOptions

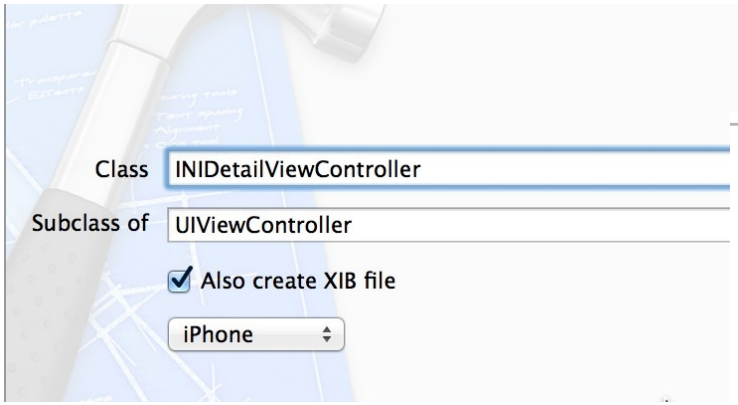
- Create a new instance of UINavigationController (navController) to replace the itemsViewController as a rootViewController for the application
- Add itemsViewController as a root controller for navController
- Add navController as a rootViewController for the application

```
13
14 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
15 {
16     self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
17     // Override point for customization after application launch.
18
19     // Create a INIItemsViewController
20     INIItemsViewController *itemsViewController = [[INIItemsViewController alloc] init];
21
22     // Place INIItemsViewController's table view in the window hierarchy
23
24     // Create an instance of a UINavigationController
25     // its stack contains only itemsViewController
26
27     UINavigationController *navController = [[UINavigationController alloc] initWithRootViewController: itemsViewController];
28
29     // Place navigation controller's view in the window hierarchy
30     self.window.rootViewController = navController;
31
32     self.window.backgroundColor = [UIColor whiteColor];
33     [self.window makeKeyAndVisible];
34     return YES;
35 }
36
```

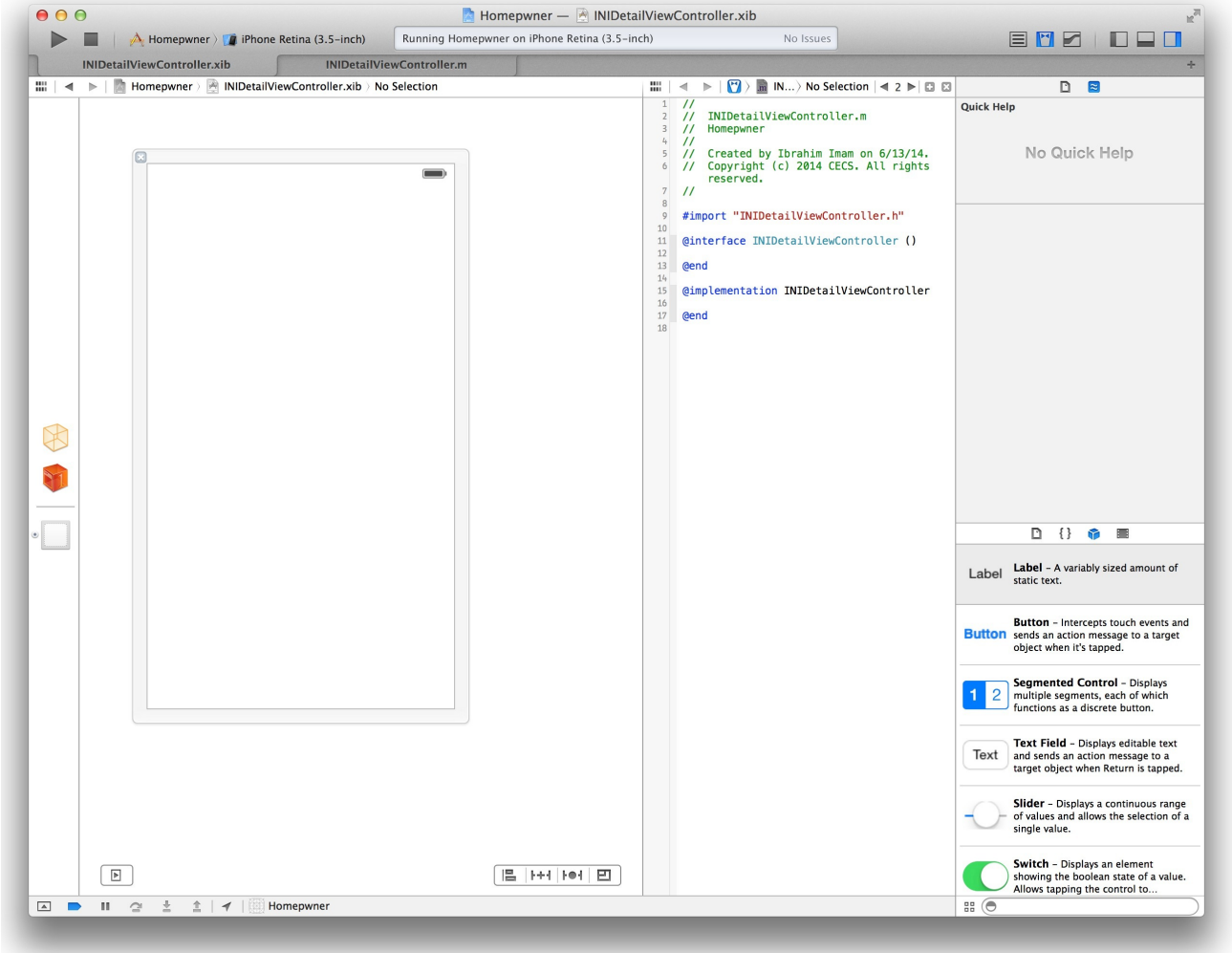


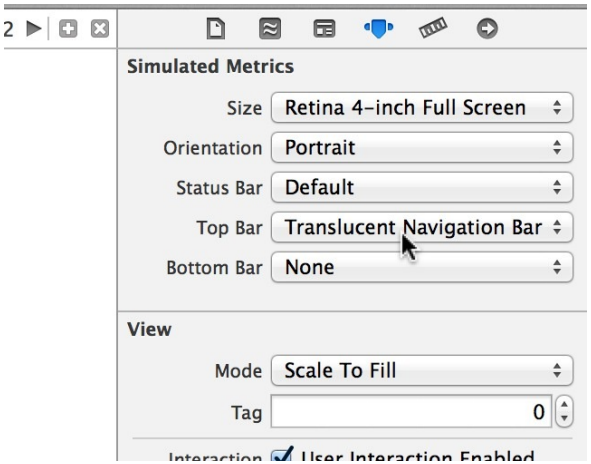
Build the details view controller

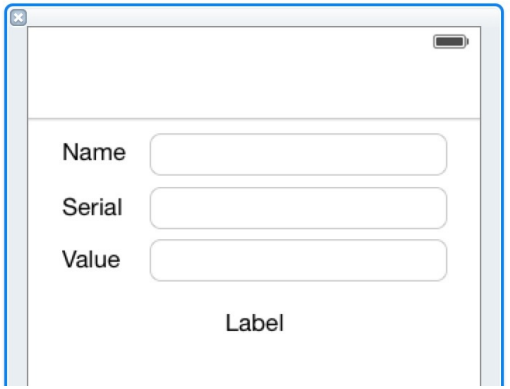
- Create a new Objective-C class called DetailViewController as a subclass of UIViewController with a XIB user interface.
- Delete all the implementation code in the implementation file keeping only the import statement and the directives.




```
1  //
2  //  INIDetailViewController.m
3  //  Homepwner
4  //
5  //  Created by Ibrahim Imam on 6/13/14.
6  //  Copyright (c) 2014 CECS. All rights reserved.
7  //
8
9  #import "INIDetailViewController.h"
10
11 @interface INIDetailViewController ()
12
13 @end
14
15 @implementation INIDetailViewController
16
17 @end
18
```

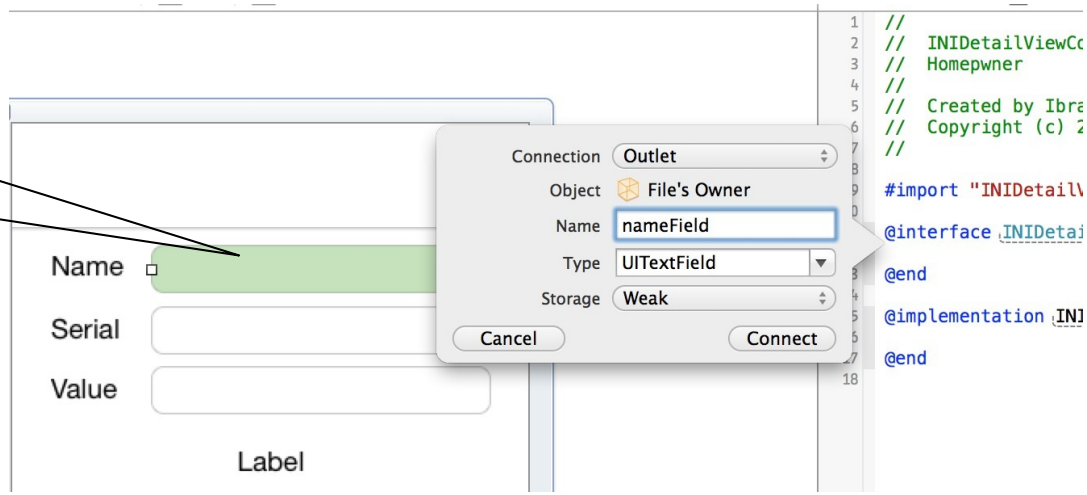

Homepwner — INIDetailViewController.xib
Homepwner › iPhone Retina (3.5-inch)
Running Homepwner on iPhone Retina (3.5-inch)
No Issues
INIDetailViewController.xib
INIDetailViewController.m
Homepwner › INIDetailViewController.xib › No Selection
IN... › No Selection
Quick Help
No Quick Help
// INIDetailViewController.m Homepwner // Created by Ibrahim Imam on 6/13/14. Copyright (c) 2014 CECS. All rights reserved.
#import "INIDetailViewController.h"
@interface INIDetailViewController ()
@end
@implementation INIDetailViewController
@end
Label – A variably sized amount of static text.
Button – Intercepts touch events and sends an action message to a target object when it's tapped.
Segmented Control – Displays multiple segments, each of which functions as a discrete button.
Text Field – Displays editable text and sends an action message to a target object when Return is tapped.
Slider – Displays a continuous range of values and allows the selection of a single value.
Switch – Displays an element showing the boolean state of a value. Allows tapping the control to...
Homepwner


Simulated Metrics
Size Retina 4-inch Full Screen
Orientation Portrait
Status Bar Default
Top Bar Translucent Navigation Bar
Bottom Bar None
View
Mode Scale To Fill
Tag 0
User Interaction Enabled


Name
Serial
Value
Label

Making the connections of the details view

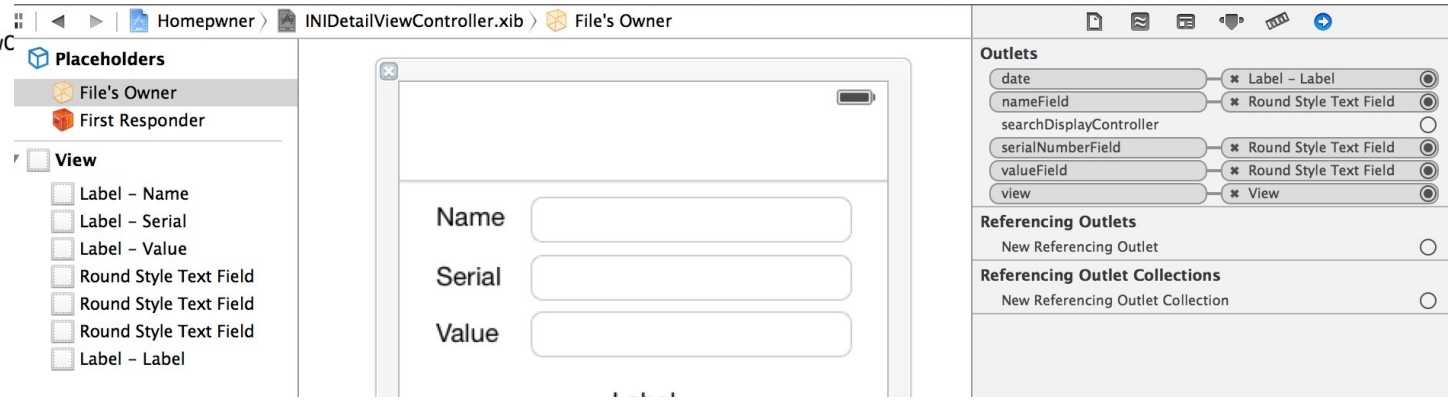
Control-Drag from the fields to the inside of the attributes brackets.

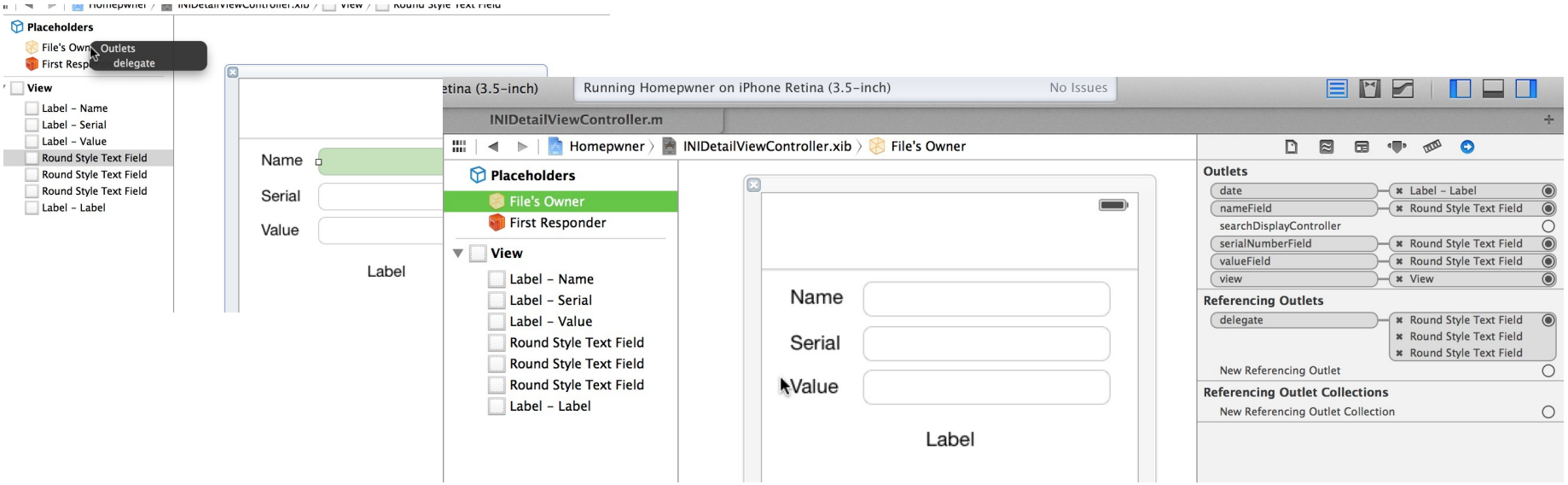


All the connections of the details view

The properties

```
#import "INIDetailViewController.h"

@interface INIDetailViewController ()
@property (weak, nonatomic) IBOutlet UITextField *nameField;
@property (weak, nonatomic) IBOutlet UITextField *serialNumberField;
@property (weak, nonatomic) IBOutlet UITextField *valueField;
@property (weak, nonatomic) IBOutlet UILabel *date;
@end

@implementation INIDetailViewC

@end
```





Pushing view controllers

- Recall that for the tab bar controller, we create all the view controllers needed by the tabs and immediately added them to the tab bar controller's viewControllers array. I think of the view controllers as peers.
- This is not the case for a UINavigationController, the view controllers are descendants of one another.

  The viewControllers array of a navigation controller is dynamic – you start with a root view controller and push view controllers depending on user input.

  Therefore, some object other than the navigation controller needs to create the instance of INIDetailViewController and be responsible for adding it to the stack.

  This object must meet two requirements:

  - it needs to know when to push a INIDetailViewController onto the stack, and
  - it needs a pointer to the navigation controller to send the navigation controller messages, namely, pushViewController: animated:.

INIItemsViewController is the pushing view controllers

- INIItemsViewController fills both requirements.

  It knows when a row is tapped in a table view because, as the table view's delegate, it receives the message "tableView:didSelectRowAtIndexPath: when this event occurs.

  Any view controller in a navigation controller's stack can get a pointer to that navigation controller by sending itself the message navigationController.

  As the root view controller, INIItemsViewController is always in the navigation controller's stack and thus can always access it.

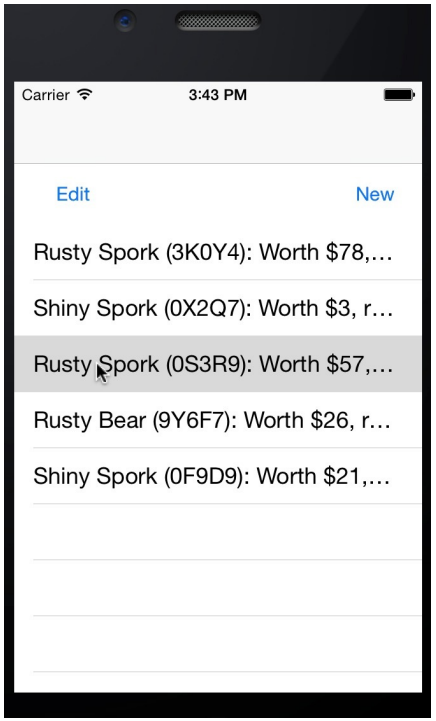
- Therefore, INIItemsViewController will be responsible for creating the instance of INIDetailViewController and adding it to the stack.

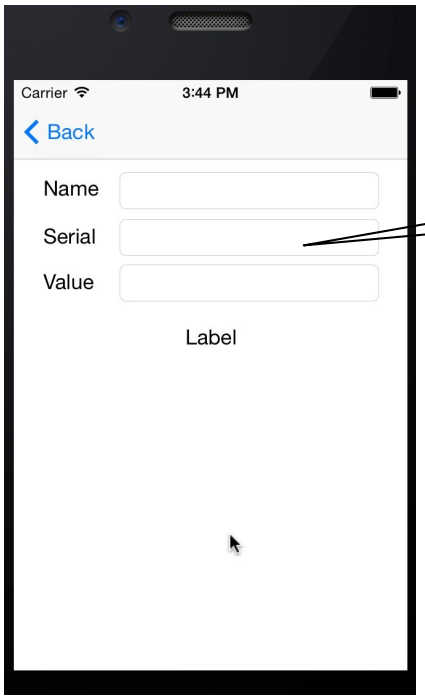
Modifying INIItemsViewController

- When a row is tapped in a table view, its delegate is sent tableView: didSelectRowAtIndexPath:, which contains the index path of the selected row.

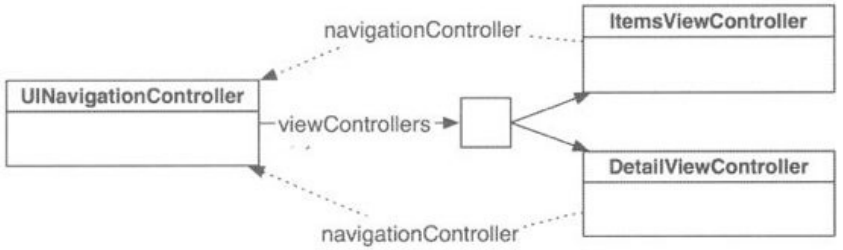
```
#import "INIItemsViewController.h"

#import "INIDetailViewController.h"
#import "INIItem.h"
#import "INIItemStore.h"

@interface INIItemsViewController ()
```

```
- ( void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:( NSIndexPath *)indexPath
{
    INIDetailViewController * detailViewController = [[INIDetailViewController alloc] init];
    // Push it onto the top of the navigation controller's stack
    [self.navigationController pushViewController: detailViewController animated: YES];
}
```





Details view and Back button but no data yet



Passing data between view controllers

1. Add INIItem instance as a property
2. As the detail view is about to appear put item's info in the appropriate fields

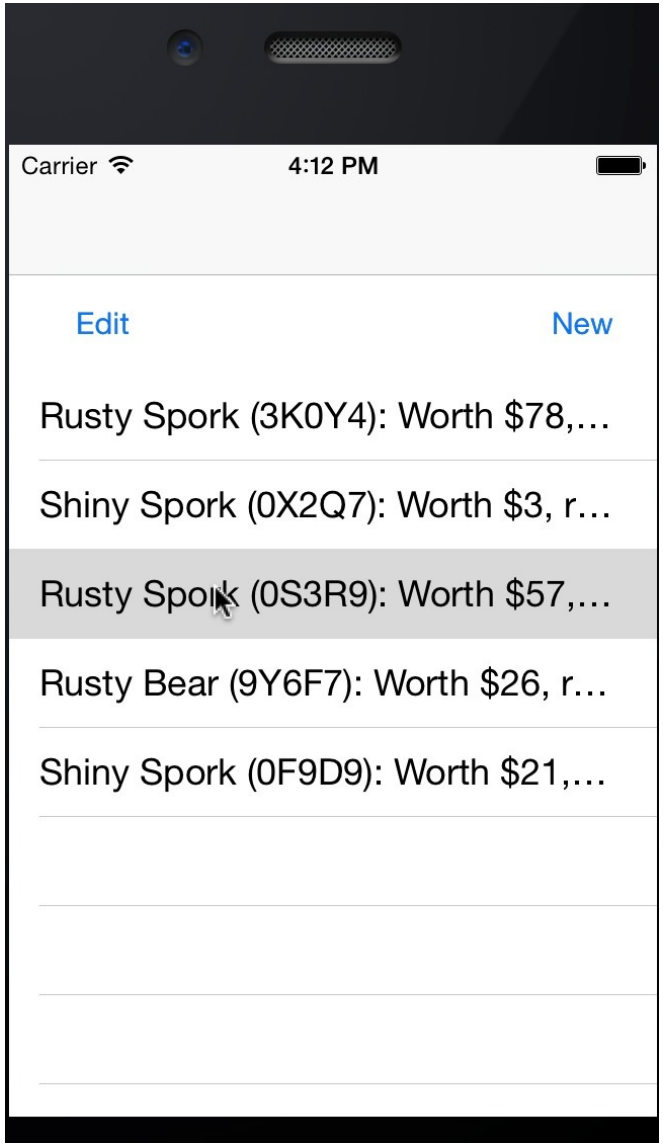
```
//

#import <UIKit/UIKit.h>

@class INIItem;

@interface INIDetailViewController : UIViewController

@property (nonatomic, strong) INIItem *item;

@end
```

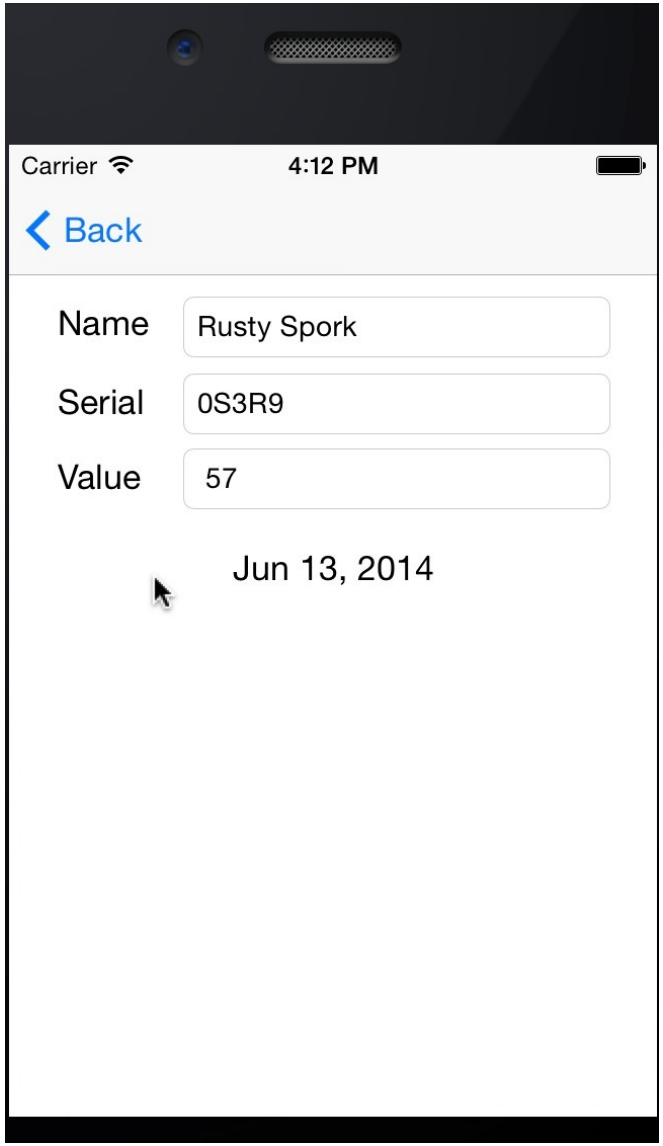
```
- (void) viewWillAppear:(BOOL) animated
{
    [super viewWillAppear: animated];
    INIItem *item = self.item;
    self.nameField.text = item.itemName;
    self.serialNumberField.text = item.serialNumber;
    self.valueField.text = [NSString stringWithFormat:@"% d", item.valueInDollars];
    // You need an NSDateFormatter that will turn a date into a simple date string static
    NSDateFormatter *dateFormatter;
    if (!dateFormatter) {
        dateFormatter = [[NSDateFormatter alloc] init];
        dateFormatter.dateStyle = NSDateFormatterMediumStyle;
        dateFormatter.timeStyle = NSDateFormatterNoStyle;
    }
    // Use filtered NSDate object to set dateLabel contents
    self.dateLabel.text = [dateFormatter stringFromDate: item.dateCreated];
}
```

Modify tableView:didSelect... to extract the appropriate item from the sharedStore and pass it to the detail view controller

```
- ( void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:( NSIndexPath *)indexPath
{
    INIDetailViewController * detailViewController = [[INIDetailViewController alloc] init];

    NSArray *items = [[INIItemStore sharedStore] allItems];
    INIItem *selectedItem = items[indexPath.row];
    // Give detail view controller a pointer to the item object in row
    detailViewController.item = selectedItem;
    // Push it onto the top of the navigation controller's stack
    [self.navigationController pushViewController: detailViewController animated: YES];
}
```

Displaying Details





Appearing and disappearing Views. (Saving changes)

- Whenever a UINavigationController is about to swap views, it sends out two messages:

  viewWillDisappear: to the UIViewController that is about to be popped off the stack

  viewWillAppear: to the UIViewController that will become the top of the stack

- As the detail view is about to disappear, it must save any changes the user has made to the instance item of INIItem (We will override viewWillDisappear: to accomplish this)
- As the table view is about to reappear it needs to refresh the data by reloading it into the table from the shared store (We will override viewWillAppear: to accomplish this).

```
32
33  - (void)viewWillDisappear:(BOOL)animated
34  {
35      [super viewWillDisappear:animated];
36
37      [[self view] endEditing:YES];
38
39      [item setItemName: [nameField text]];
40      [item setSerialNumber: [serialNumberField text]];
41      [item setValueInDollars: [[valueField text] intValue]];
42  }
43
```

```
127
128 - (void) viewWillAppear:(BOOL)animated
129 {
130     [super viewWillAppear: animated];
131     [[self tableView] reloadData];
132 }
133
```

Saving changes edit changes

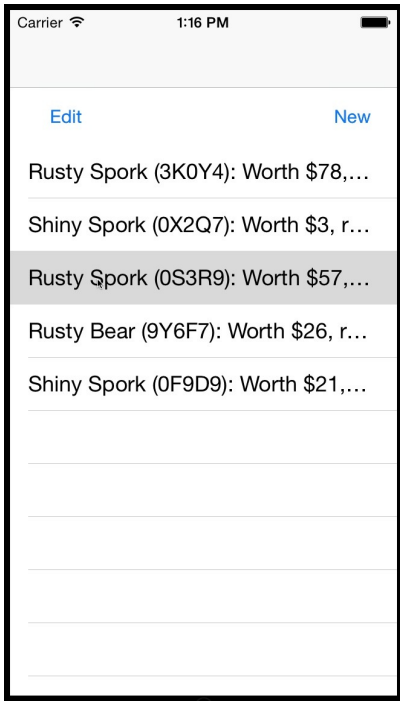
- As the detail view is about to disappear, it must save any changes the user has made to the instance item of INIItem
- We will override viewWillDisappear: to accomplish this

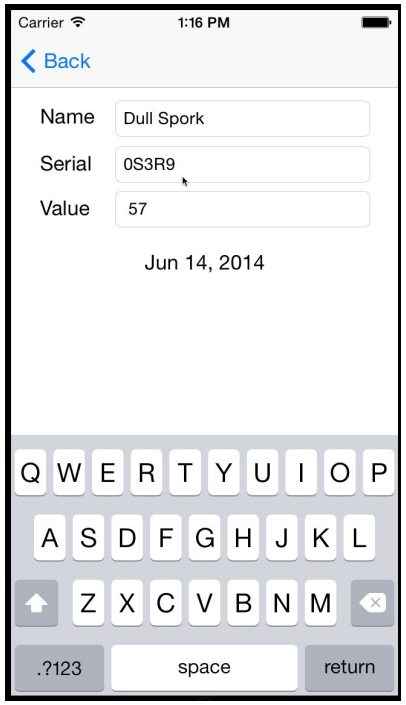
```
39
40  - (void) viewWillDisappear:(BOOL)animated
41  {
42      [super viewWillDisappear: animated];
43
44      // Clear first responder and retract any keyboard
45      [self. view endEditing: YES];
46
47      // " Save" changes to item
48      INIItem *item = self.item;
49      item.itemName = self.nameField.text;
50      item.serialNumber = self.serialNumberField.text;
51      item.valueInDollars = [self.valueField.text intValue];
52  }
```

Refreshing table view to reflect edit changes

- As the table view is about to reappear it needs to refresh the data by reloading it into the table from the shared store
- We will override viewWillAppear: to accomplish this.

```
- (void) viewWillAppear:( BOOL) animated
{
    [super viewWillAppear: animated];
    [self.tableView reloadData];
}
```





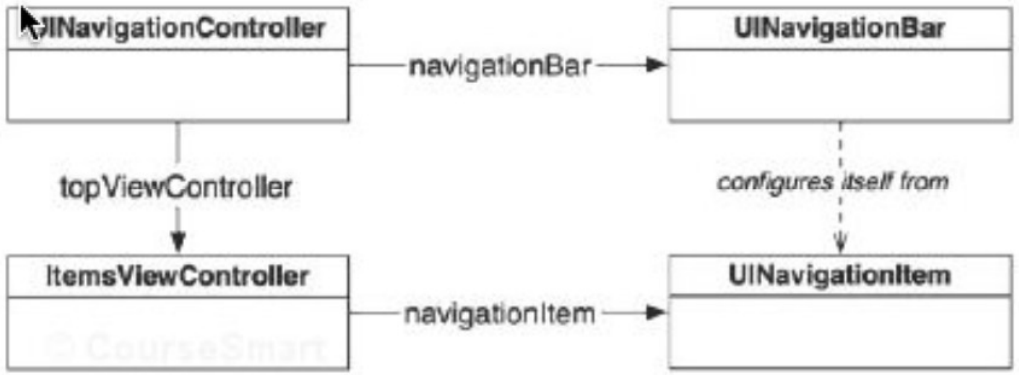
UINavigationBar and UINavigationItem

- The UINavigationBar class implements a control for navigating hierarchical content. It typically contains buttons for navigating up and down a hierarchy. The primary properties are:

  A left (back) button,

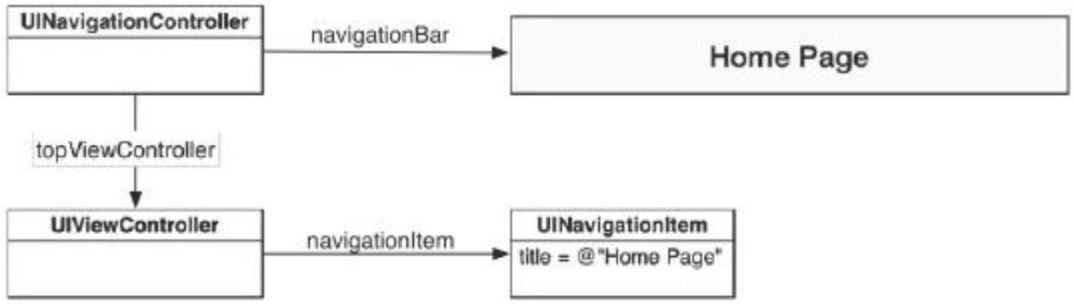
  A center title,

  An optional right button.
- The UINavigationItem is not a subclass of UIView
- The navigation item supplies the navigation bar with the content it needs to draw.



UINavigationBar and UINavigationItem (cont.)

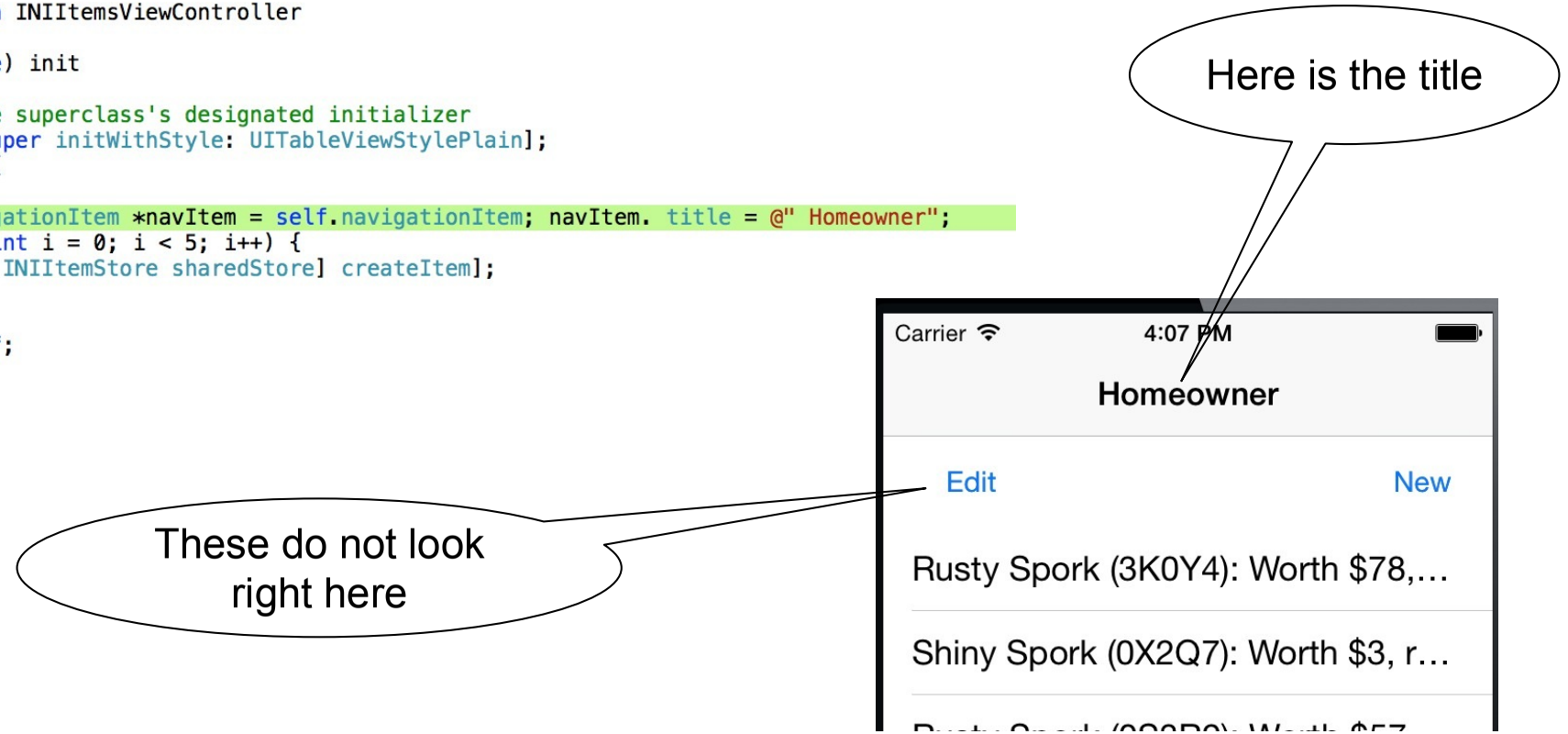
- When a UIViewController comes to the top of the stack of a UINavigationController, the UINavigationBar uses the UIViewController's navigationItem to configure itself
- The UINavigationItem class encapsulates information about a navigation item pushed on a UINavigationBar object's stack.
- It specifies what is displayed on the navigation bar when it is the top item and also how it is represented when it is the back item



Setting the title to “Homeowner”

- In INIItemsViewController.m we need to modify init to set the navigationItem’s title to read Homopwner.

```
@implementation INIItemsViewController

- (instancetype) init
{
    // Call the superclass's designated initializer
    self = [ super initWithStyle: UITableViewStylePlain];
    if (self) {

        UINavigationItem *navItem = self.navigationItem; navItem. title = @" Homeowner";
        for ( int i = 0; i < 5; i++) {
            [[ INIItemStore sharedStore] createItem];
        }
    }
    return self;
}
```

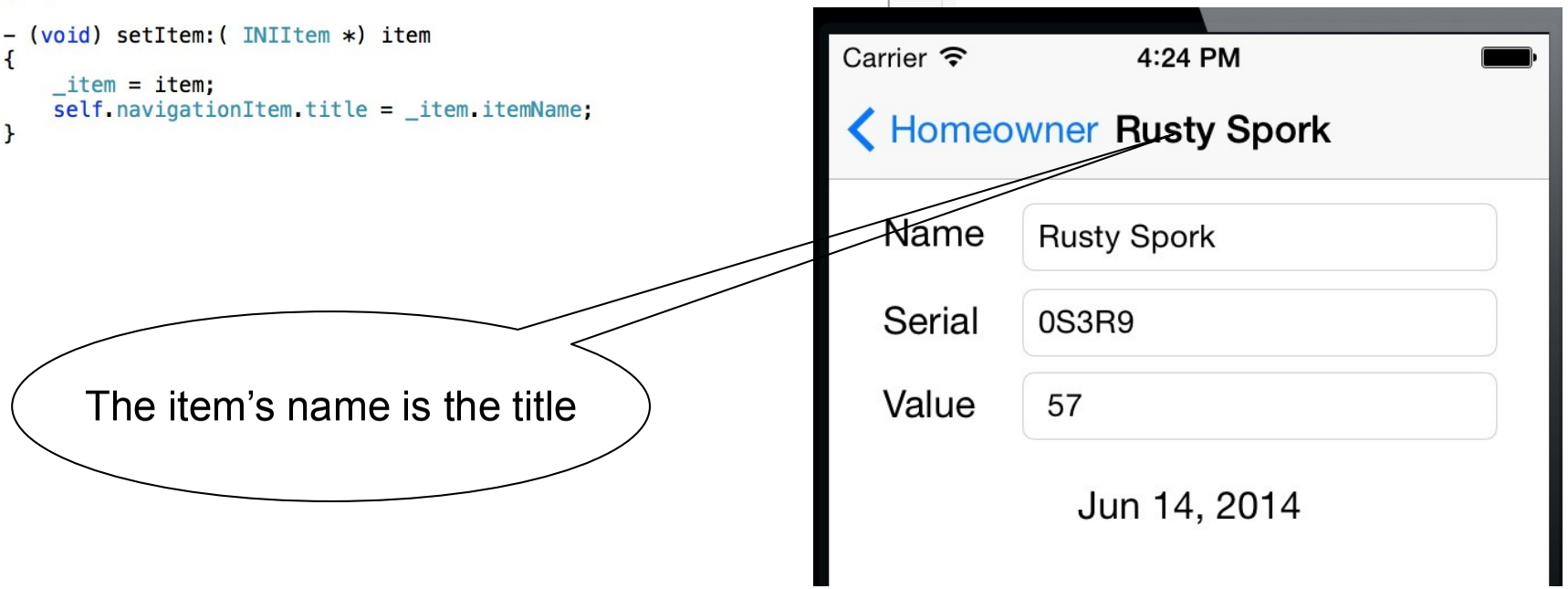


Setting title for the detail navigation bar

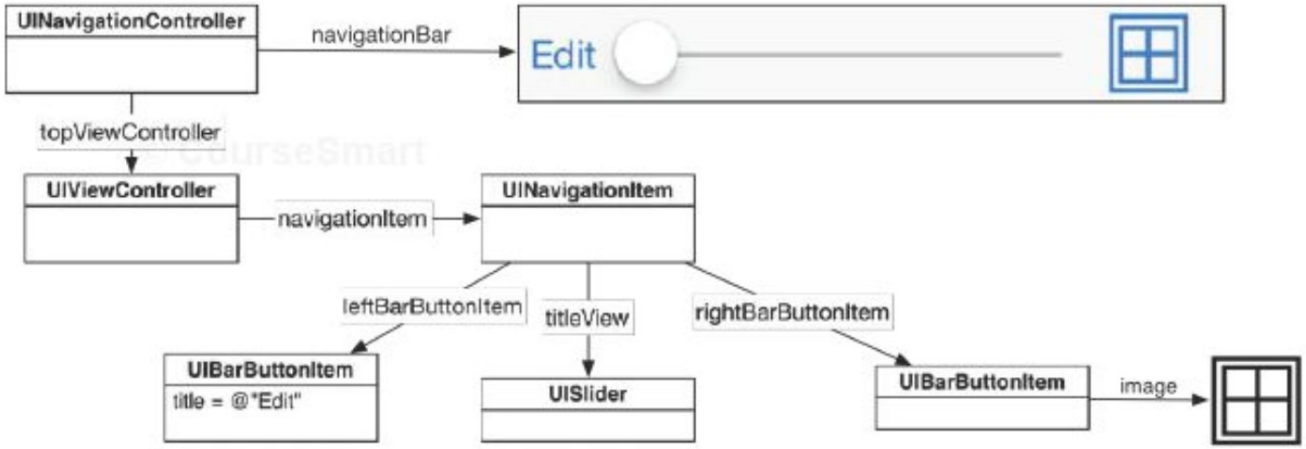
- Override the setItem: for the property item in the INIDetailViewController so that it uses the name of the item as the title of the navigation bar of the detail view

```
11
12 @interface INIDetailViewController ()
13 @property (weak, nonatomic) IBOutlet UITextField *nameField;
14 @property (weak, nonatomic) IBOutlet UITextField *serialNumberField;
15 @property (weak, nonatomic) IBOutlet UITextField *valueField;
16 @property (weak, nonatomic) IBOutlet UILabel *dateLabel;
17 @end
18
19 @implementation INIDetailViewController
20
21 - (void) setItem:( INIItem *) item
22 {
23     _item = item;
24     self.navigationItem.title = _item.itemName;
25 }
26
```

```
11 @class INIItem;
12
13 @interface INIDetailViewController : UIViewController
14
15 @property (nonatomic, strong) INIItem *item;
16
17 @end
18
```



UINavigationItem with everything

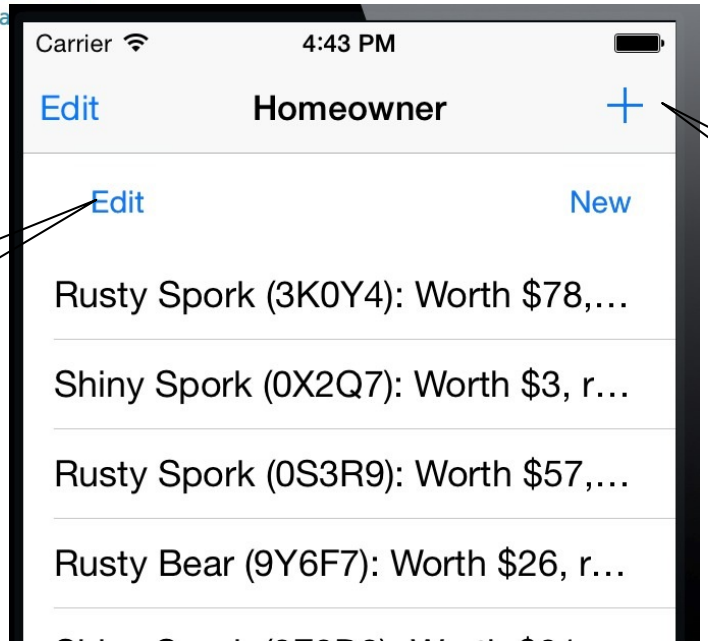


- UINavigationItem is not a view, UINavigationItem encapsulates information that UINavigationBar uses to configure itself.
- Also, UIBarButtonItem is not a view, but holds the information about how a single button on the UINavigationBar should be displayed.
- The third customizable area of a UINavigationItem is its titleView. You can either use a basic string as the title or have a subclass of UIView sit in the center of the navigation item.

Final Changes

- We need to get rid of the “edit” and “new” buttons’ view and replace then with a and “edit” and “+” buttons on the items view’s navigation bar

```
- (instancetype) init
{
    // Call the superclass's designated initializer
    self = [ super initWithStyle: UITableViewStylePlain];
    if (self) {

        UINavigationItem *navItem = self.navigationItem; navItem. title = @" Homeowner";

        // Create a new bar button item that will send
        // addNewItem: to INIItemsViewController
        UIBarButtonItem *bbi = [[UIBarButtonItem alloc] initWithBarButtonSystemItem: UIBarButtonSystemItemAdd target: self action:@selector(addNewItem:)];
        // Set this bar button item as the right item in the navigationItem
        navItem.rightBarButtonItem = bbi;

        navItem.leftBarButtonItem = self. editButtonItem;

        for ( int i = 0; i < 5; i++) {
            [[ INIItemStore sharedStore] crea
        }
    }
    return self;
}
```



These must go

This is what we want

Cleaning up

- We need to get rid of the “edit” and “new” buttons’ view

Delete the highlighted stuff from NIItemsViewController.
You can also delete the declaration of the headerView property.
You can also remove the file HeaderView.xib from the project navigator.

```
- (IBAction) toggleEditingMode:(id) sender
{
    // If you are currently in editing mode...
    if (self.isEditing) {

        // Change text of button to inform user of state
        [sender setTitle: @"Edit" forState: UIControlStateNormal];

        // Turn off editing mode
        [self setEditing: NO animated: YES];
    } else {

        // Change text of button to inform user of state
        [sender setTitle:@"Done" forState: UIControlStateNormal];
        // Enter editing mode
        [self setEditing: YES animated: YES];
    }
}

- (UIView *) headerView
{
    // If you have not loaded the headerView yet...
    if (!_headerView) {
        // Load HeaderView.xib
        [[ NSBundle mainBundle] loadNibNamed:@"HeaderView" owner: self options: nil];
    }
    return _headerView;
}
```

```
- (void) viewDidLoad
{
    [super viewDidLoad];
    [self.tableView registerClass:[UITableViewCell class] forCellReuseIde

    UIView *header = self.headerView;
    [self.tableView setTableHeaderView: header];
}
```

You must keep this

```
(IBAction) addNewItem:(id) sender
{
    // NSInteger lastRow = [[self tableView] numberOfRowsInSection: 0];  // This will crash the app
    // Create a new BNRItem and add it to the store

    INIItem *newItem = [[INIItemStore sharedStore] createItem];

    // Figure out where that item is in the array
    NSInteger lastRow = [[[INIItemStore sharedStore] allItems] indexOfObject: newItem];
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow: lastRow inSection: 0];

    // Insert this new row into the table
    [self. tableView insertRowsAtIndexPaths:@[indexPath] withRowAnimation: UITableViewRowAnimationTop];
```