ES3 Lecture 5

InterfaceBuilder and UI development

Creating views programmatically

- The loadView method of a UIViewController subclass creates the view for the view controller
- Can add in UI components here
  - You have to manually specify properties like color, position, etc.

```
- (void) loadView {
    //Full size
    self.view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];

    UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(50, 50, 100, 20)];
    label.text = @"Test";
    [self.view addSubview:label];

    [label release]; // the view will hold onto a reference now
}
```

  - Remember, you must add the control to the **view** of the viewcontroller
- This can be very verbose
  - But some people prefer it because it involves less "*magic*"

Using InterfaceBuilder

- InterfaceBuilder is a visual editor for iPhone applications
  - Allows you to quickly and easily add and arrange views
  - These views are linked to objects in your code
- If you're making an app with conventional GUI components, InterfaceBuilder is the way to go
- Can seem a bit like magic
  - Connections between objects are not visible in source
  - NIB files store archived objects which store this data
- Once you get used to it, it's very powerful

InteFaceBuilder

- IB can create objects and store them in archives (NIB files)
- These objects can be connected to your code in several ways

  - **Outlet**: a reference to an IB object in your code
    - e.g. so you can set the text of a label

  - **Action**: a message that an IB object can send to your code
    - e.g. a message to be sent when a button is pushed

  - And other miscellaneous ways...
    - **Delegate**: an object in your code which is a delegate for an IB object
      - e.g. responding to actions on a table

    - **Data source**: an object that provides data for a view
      - e.g. a table data source can provide row entries when requested

NIB files

- NIB files (actually extension xib) store data from InterfaceBuilder

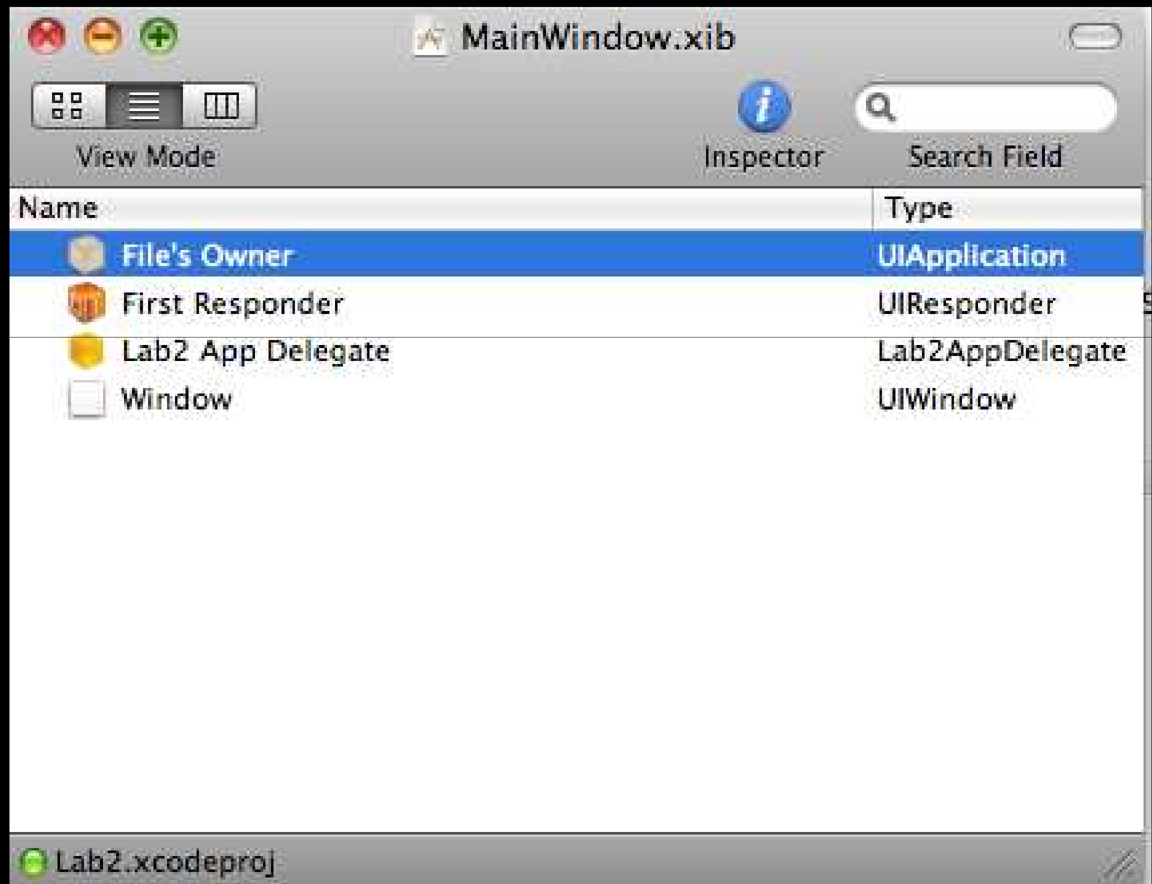
- You can open them in XCode, InterfaceBuilder will be launched
- When creating new UIViewController subclasses, XCode can create a blank NIB file and use it to create the viewController

```
UIViewController *controller = [[UIViewController alloc]
initWithNibName:@"mainview.xib"];
```

- **initWithNibName** initialises a viewcontroller and links it to an NIB file

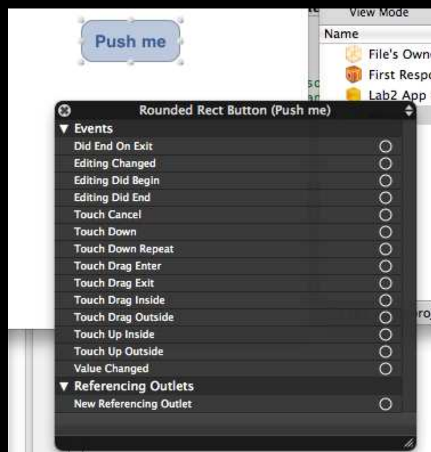
- **mainwindow.xib** is automatically created as and linked to the main UIApplication

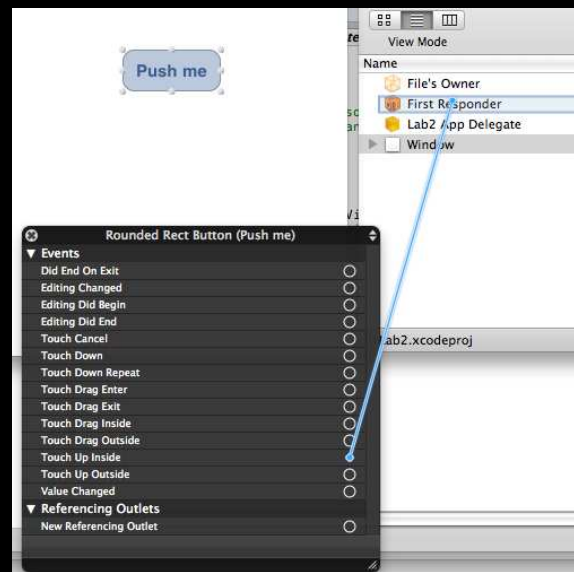
MainWindow.xib
View Mode
Inspector
Search Field
File's Owner
First Responder
Lab2 App Del...
Window
Lab2.xcodeproj

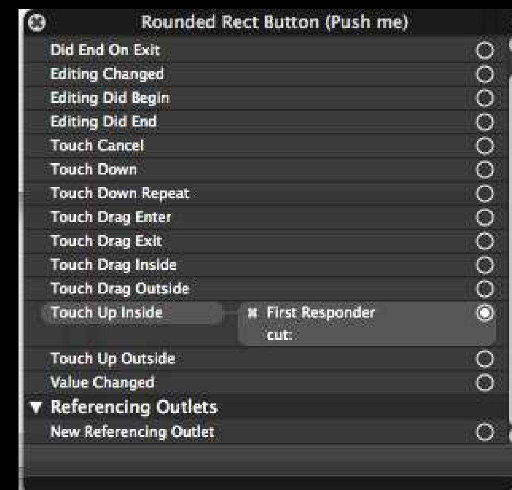

MainWindow.xib
View Mode
Inspector
Search Field
Name
Type
File's Owner
UIApplication
First Responder
UIResponder
Lab2 App Delegate
Lab2AppDelegate
Window
UIWindow
Lab2.xcodeproj

Using InterfaceBuilder (II)

- Right-click (or control-click) to bring up the list of connections an object or UI component can make
  - drag to another object
  - a list of possible connections appear
  - click on the connection you want to make
    - If a connection doesn't appear the receiving object is not of the right type, or doesn't have methods or variables of the right type

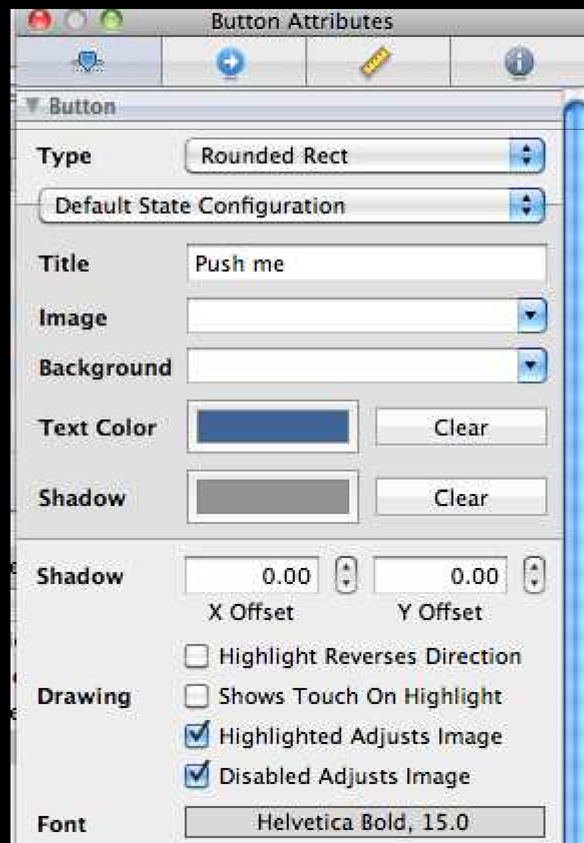


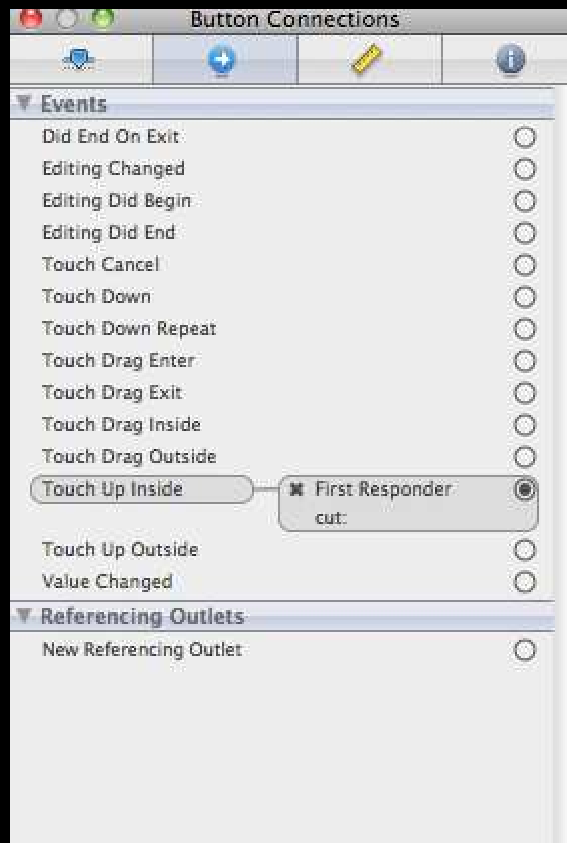


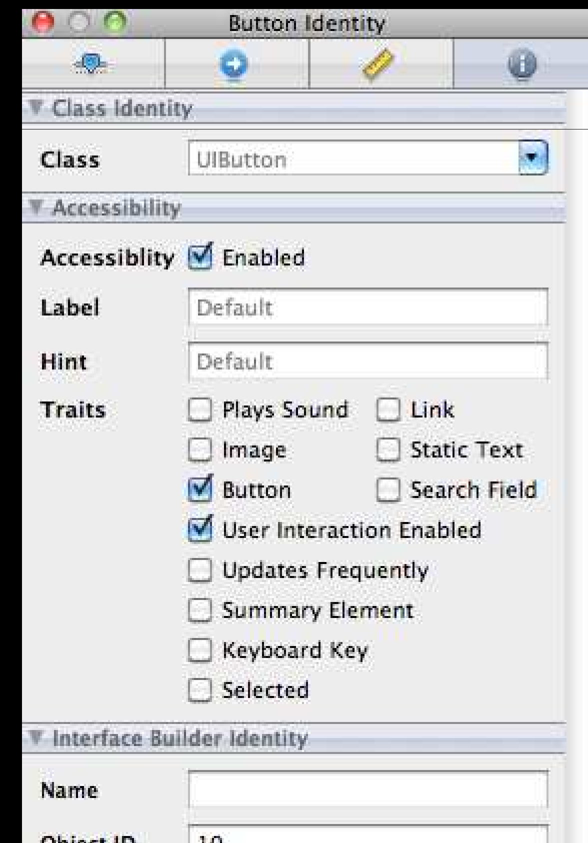


Using InterfaceBuilder (II)

- The inspector lets you set properties of objects and check connections
  - it has four panes -- third pane is just size information







Inspector panes

- The first pane (attributes) allows things like font, color, and other appearance attributes to edited without using any code
- The second (connections) shows a permanently available record of connections to and from this object
- The third has size and alignment options
- The fourth (identity) allows the class of the object to be set and various underlying attributes of the object to be accessed

IBAction and IBOutlet

- You can link instance variables and methods between your code and InterfaceBuilder

- Mark a method as an "action receiver" by making it of return type **IBAction**
  - InterfaceBuilder will then recognize it as a valid receiver of actions
  - you will be able to create connections from actions to an object with methods returning **IBAction**

```
- (IBAction) buttonPushed {
 // do something with a button push
}
```

- To link an object to an object in InterfaceBuilder, mark it's **property** with IBOutlet

```
 UILabel *myLabel;
 }

 @property (nonatomic, retain) IBOutlet UILabel *label;
```

IBOutlet

- You can then link a user interface component *of that type* in InterfaceBuilder

- **Note: Do not instantiate that object!**
  - **InterfaceBuilder creates an instance of the object and stores it in the NIB file**

- All linking the object to the interface component does is give you a reference that you can manipulate
  - e.g. setting the text of a label, reading the value of a slider

```
- (IBAction) buttonPushed
{
  myLabel.text = [NSString stringWithFormat:@"%f", mySlider.value];
}
```

Protocols

- InterfaceBuilder requires strict adherence to protocols

- IB checks whether objects are of the right type or conform to the protocol before it allows you to even create a connection
  - if a method is not marked IBAction, it won't appear as an option when connecting actions

  - if an outlet is not of the right type, a referencing connection cannot be made (it just won't appear)

  - components which have **delegate**s or **dataSource**s (like **UITableView)** must be linked to objects which conform to the appropriate protocol
    - **UITableViewDelegate, UITableViewDataSource, for example**

UIViewController

- Combines a **UIView** and **UIResponder**
  - **UIView** is the **view** property
  - UIViewController Inherits from **UIResponder**
    - **UIResponder** handles events (i.e. touches)

- **touchesBegan** message
  - indicates a finger has gone down

- **touchesEnded** message
  - indicates a finger has been lifted

- It is very common to use a subclass of **UViewController** as a "whole screen"
  - adding controls to the viewcontrollers view

UIViewController

- **loadView**
  - this is where construction of subviews should go
    - e.g. populating a form with buttons

- **viewDidLoad**
  - this message is sent after the view is loaded
  - other initialisation should go here
  - this is where initialisation should go if your are using InterfaceBuilder

- **UIViewControllers** also respond to screen rotation changes
  - portrait <-> landscape
  - see the API docs

Handling events

- Use target action
  - Tell a component which object to send messages to
  - Note that you need to specify what events you want to listen to:
    - UIControlEvent*
    - UIControlEventTouchUpInside / DownInside most common
    - also for drags, event changes, editing start and stopping

- This will send the message **doSomethingInteresting** to **self** (the view controller object in this case) when the user touches down then up on a button:

```
//Inside view controller's loadView
UIButton *pressMe = [[UIButton alloc] initWithFrame:CGRectMake(10,10,50,50)];
[pressMe addTarget:self action:@selector(doSomethingInteresting)
forControlEvents:UIControlEventTouchUpInside];
```

Responding to a button push

```
- (void) loadView
{

...

//Inside view controller's loadView
//pressMe is UIButton instance variable of this class
pressMe = [[UIButton alloc] initWithFrame:CGRectMake(10,10,50,50)];
[pressMe addTarget:self action:@selector(doSomethingInteresting)
forControlEvents:UIControlEventTouchUpInside];
}


- (void) doSomethingInteresting:(id)sender  {
  if(sender==pressMe) {
    doSomethingElse(); // doSomethingElse is called when finger goes up inside the
button
  }

}
```

The tag property

- Every UIView (include UIButton etc.) has a tag property
- This is a NSInteger which you can set so as to identify the control or group

```
- (void) doSomethingInteresting:(id)sender  {

  if([(UIView*)sender.tag intValue] == 60) {
    doSomethingElse(); // doSomethingElse is called when finger goes up inside any
control tagged with 60
  }

}
```

Text Entry

- **UITextField** is the basic text field component

- Text entry widgets need configuration on the iPhone
  - Different types of entry (numbers, words)
  - Different actions on finishing entering text

- You must also specify a delegate for the textfield object
  - This, at a minimum, must specify what to do when the return key is pressed
  - **If you don't set this, the virtual keyboard will never be dismissed!**

- You can set the type of entry in InterfaceBuilder
  - numeric or alphabetic
  - whether or not autocorrect is enabled
  - what capitalization rules to apply (first, never, all)

UITextFieldDelegate

```
//assume there is an instance variable myTextField

//self should conform to the UITextFieldDelegate protocol
myTextField.delegate = self; // you can also do this in InterfaceBuilder

- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
  [textField resignFirstResponder]; // ESSENTIAL -- this dismisses the keyboard!
  // do something with the text...
  return YES;
}


// You can validate text by implementing this method

- (BOOL)textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range replacementString:(NSString *)string
{
   if([self isValid:replaceString])
     return YES;
   else
     return NO;

  // can also directly correct the string in this method
}
```

Animation effects

- Easy support for animation effects
  - Key part of "flashy iPhone effect"
- Views (any UI component)... can be dynamically altered
  - Alpha
  - Coloring
  - Transform (rotation, position, scale)
- You simply set a target state and specify:
  - how long it will take
  - what curve to use (just linear, or with ease in/ease out)
  - whether it repeats
  - and a target/action to use when the animation finishes
- Then commit the animation and it will play in the background
  - Notified when it is complete

Animation effects

- Animations are done with **class methods** of UIView
  - Begin an animation with **[UIView beginAnimation]**
    - need to give a unique name, and a context -- usually nil for the context will do
  - End it with **[UIView commitAnimation]**
  - Specify all parameters between these two calls!

- Important things you can set
  - duration **[UIVIew setAnimationDuration:(double)d]** in seconds
  - repeats **[UIView setAnimationRepeatCount:(int)repeats]**
  - autoreversing **[UIView setAnimationRepeatAutoreverses:(BOOL)doesReverse]**
  - curve **[UIView setAnimationCurve:curveType]**
    - Interesting -- can have simple linear transitions, or ones which "ease in" or "ease out" or both
    - i.e. gradually accelerate or decelelate

  - when it happens **[UIView setAnimationStartDate:(NSDate) when]** or **[UIView setAnimationDelay:(double)seconds]**
    - default is to start immediately

Animation delgate

- Often need to tell when an animation stops or starts
  - Set a delegate inside the begin/commit block
  - set selectors for **setAnimationWillStartSelector / setAnimationWillStopSelector**

```
[UIView beginAnimations:@"myAnimation context:nil];

[UIView setAnimationDelegate:self];
[UIView setAnimationWillStopSelector:@selector(animationOver)];

[UIView commitAnimations];

// later...

- (void) animationOver
{
  // do something interesting, like reseting the
  // properties of the view

}
```

Example: A table view

- Tables are very common in iPhone applications
  - Often used full screen

- Tables can have text, images and controls
- Each cell is a **view**
  - **i.e. it can have contain other controls**

- Each table needs a delegate and a data source
  - **delegate**: messages relating to actions of tables are sent here
  - **dataSource**: this must provide the cells that populate the table

- NB cells are requested from the **dataSource** *as needed*
- *Must link the table to a delegate and dataSource, either in InterfaceBuilder or in your code*
  - Without a dataSource nothing will be drawn!

UITableViewDataSource

- Essential part of table construction
- Supplies data about the table
  - Number of sections
  - Number of rows in a section
  - The **cells** for each row
    - Note: cells are not strings, but general **views**
  - **A pool of cell objects is maintained and reused**
    - don't need to worry about this if you subclass UITableViewController in XCode

- **need to implement:**
  - **numberOfSectionsInTableView** -- return number of sections
  - **sectionTitleIndexTitlesForTableView --** return array of titles
  - **numberOfRowsInSection --** given a section number, return no. of rows
  - **cellForRowAtIndexPath --** return a cell given an index path (row + section)

UITableViewDelegate

- Responds to user interaction with the table
  - key messages
    - **didSelectRowAtIndexPath**
    - **didEndEditingRowAtIndexPath**
  - Sent when user selects or edits row respectively

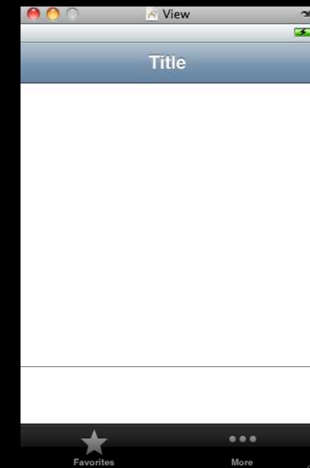
- Uses **NSIndexPath**
  - Represents a **section** and a **row**
  - Can be read from the **section** and **row** properties of the NSIndexPath instance
  - Construct with **[NSIndexPath indexPathForRow:row inSection:section]**

UITableViewController

- Combines a **UITableView** with **delegate** and **dataSource**
- Simple, all-in-one solution for fullscreen table views
    - Not very flexible, unsuitable for tables which are part of a larger interface
- Just override the necessary methods
    - XCode fills them in for you when you create a subclass of **UITableViewController**

UINavigationController

- Makes managing multi-page hierarchical views easy
- You can "push" a view onto a UINavigationController
  - and pop it off again



- Controller automatically creates a back button at top of screen
  - will automatically pop the view when pressed
- Use: create **UINavigationController** instance with a root view controller
  - this is view controller for the "base" page of the controller
- When you want to replace it with a new view, call **pushViewController**

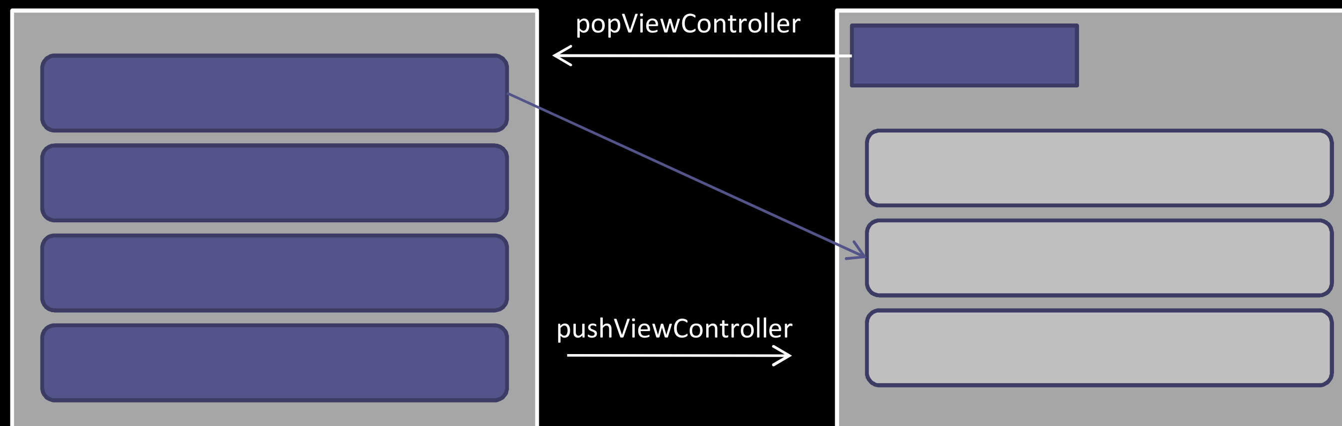
```
UINavigationController *navigationController = [[UINavigationController alloc]
initWithRootViewController:baseViewController];

// later

[navigationController pushViewController:newDisplay animated:YES];
```

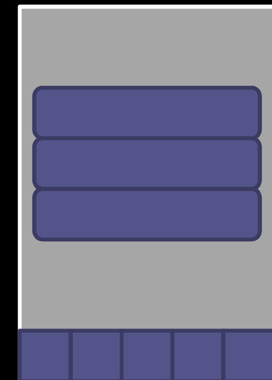
UINavigationController

- Can jump back to the root page by calling **popToRootViewControllerAnimated**
- Navigation bar visibility can be toggled with **setNavigationBarHidden**
- **UINavigationController** is excellent for navigation in conjunction UITableView
  - When the user selects a table element, drill down, and push on a new view with new more detailed table



UITabBarController

- An analogue of **UINavigationContoller**, but for for short linear lists, not hierarchies
  - (optionally) allows user to rearrange tab bar
- Displays a set of page tabs at the bottom
  - Each is linked to a view
- You can link them in code, or directly in InterfaceBuilder (using the Inspector)
  - Just create, and pass an **NSArray** of **UIViewControllers**
  - These will appear as tabs
- Has a **UITabBar** property, which in turn takes an array of **UITabBarItems**
  - Each tab bar item can have a title and an image
  - image should be 30x30 pixels



MapView

- Map view is a ready made widget for display maps
  - Provided by **MKMapView**

- **Many handy features**
  - Can center on current location (or other specified location)
  - Add, remove and select annotations (i.e. pins on the map)
  - Convert screen co-ordinates to and from world co-ordinates

  - Optionally allow user to zoom and pan the map

- Part of the **MapKit** framework -- you need to add it to your project, and import <MapKit/MapKit.h>

Using MapView

- Create an instance of MKMapView with the frame it's to go in
  - add it to your view controller

```
- (void) viewDidLoad { mapView = [[MKMapView alloc]
initWithFrame:self.view.bounds];
[self.view addSubview:mapView];
}
```

  - Show current location by setting **showsUserLocation** to **YES**
  - Drop a pin by using **addAnnotation**with a **MKPinAnnotationView** instance
    - Can make more complex annotations by subclassing **MKAnnotation**
  - Recenter with **setCenterCoordinate** or **setRegion** to set a visible region (sets zoom level too, based on a lat/long span)

```
//drops a pin somewhere near Glasgow...
self.mapView.showsUserLocation = YES;
MKPinAnnotationView *annotation = [[MKPinAnnotation alloc] init];
annotation.coordinate.latitude = 55.0;
annotation.coordinate.longitude = -4.0;
[self.mapView addAnnoation:annotation];
```

Dealing with Multitouch

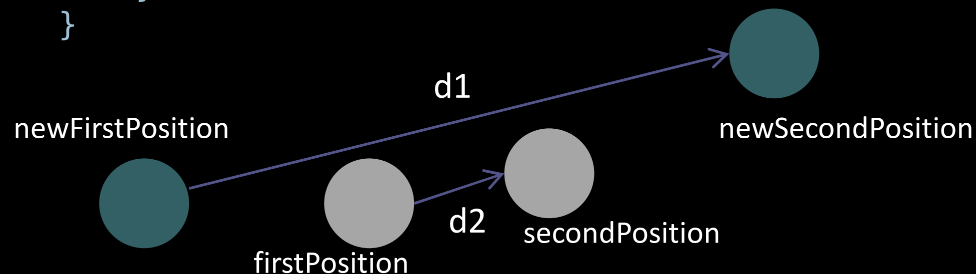
- Lots of iPhone applications make a big deal of multitouch
  - But there aren't any built in methods to deal with multitouch interaction
- You get **touchesBegan**, **touchesMoved** and **touchesEnded** calls
  - with a set of touches
    - assuming you set the view to respond to multitouch!
- Assume we want to make a stretch to scale gesture, *a la* the image viewer
- We need to record initial finger positions **when both fingers are down**
  - when they move, we need to update the scale appropriately
  - *note: UITouch objects persist throughout a multitouch interaction*
    - no longer, but that's enough...

Simple stretch gesture

```
// in the class definition
UITouch *firstTouch, *secondTouch;
CGPoint firstPosition, secondPosition;


- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    // get the touches on this control
    NSSet *touches = [event touchesForView];

    // only if we now have two touches (we ignore the first touchesBegan, where
the first finger went down)
    if(touches.count==2)   {
          NSEnumerator *touchEnumerator = [touches objectEnumerator];
          firstTouch = [touchEnumerator nextObject];
          secondTouch = [touchEnumerator nextObject];
          firstPosition = [firstTouch  locationInView:self];
          secondPosition = [secondTouch locationInView:self];
    }
}
```



Simple stretch gesture

```
- (void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
 // get the touches on this control
    NSSet *touches = [event touchesForView];

    // if we have two touches, we must have had two-finger touchesBegan before
    if(touches.count==2)   {
          CGPoint newFirstPosition, newSecondPosition;
          newFirstPosition = [firstTouch locationInView:self];
          newSecondPosition = [secondTouch locationInView:self];

          double dx1,dx2,dy1,dy2,d1,d2;
          // compute distance
          dx1 = newFirstPosition.x - newSecondPosition.x;
          dx2 = firstPosition.x - secondPosition.x;
          dy1 = newFirstPosition.y - newSecondPosition.y;
          dy2 = firstPosition.y - secondPosition.y;
          d1 = sqrt(dx1*dx1+dy1*dy1);
          d2 = sqrt(dx2*dx2+dy2*dy2);

          // new scale is ratio of distances...
          self.scale = d1/d2;

    }

}
```

Summary

- InterfaceBuilder provides a way of visually connecting objects
  - actions send messages, marked **IBAction**
  - outlets make references to objects, marked **IBOutlet**
- Table views provide powerful table support
  - read data as needed from a dataSource
- Navigation controllers make it easy to manipulate a stack of "pages"
  - great for hierarchical viewing
- Tab bars provide simple multi-page interaction
- MapKit provides a ready made map viewer and annotation tool