Xamarin University

IOS205

Navigation Patterns

Download class materials from
university.xamarin.com

Microsoft

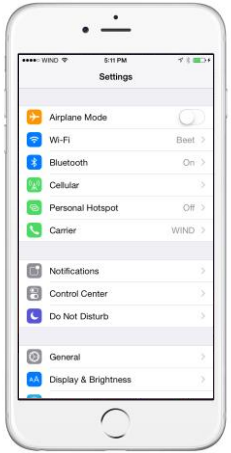
Xamarin University

Objectives

1. Progress through pages of data with stack-based navigation
2. Show different views of related data with tab navigation
3. Display hierarchical relationships with master/detail navigation

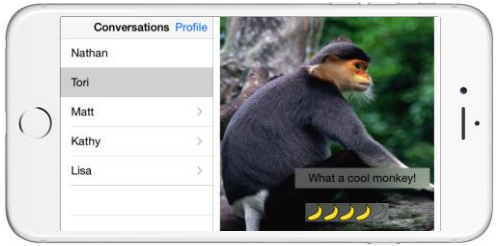
Navigation patterns

- iOS provides several ways to structure navigation in your application – must decide the most effective way to present your information



Stack

Tabs



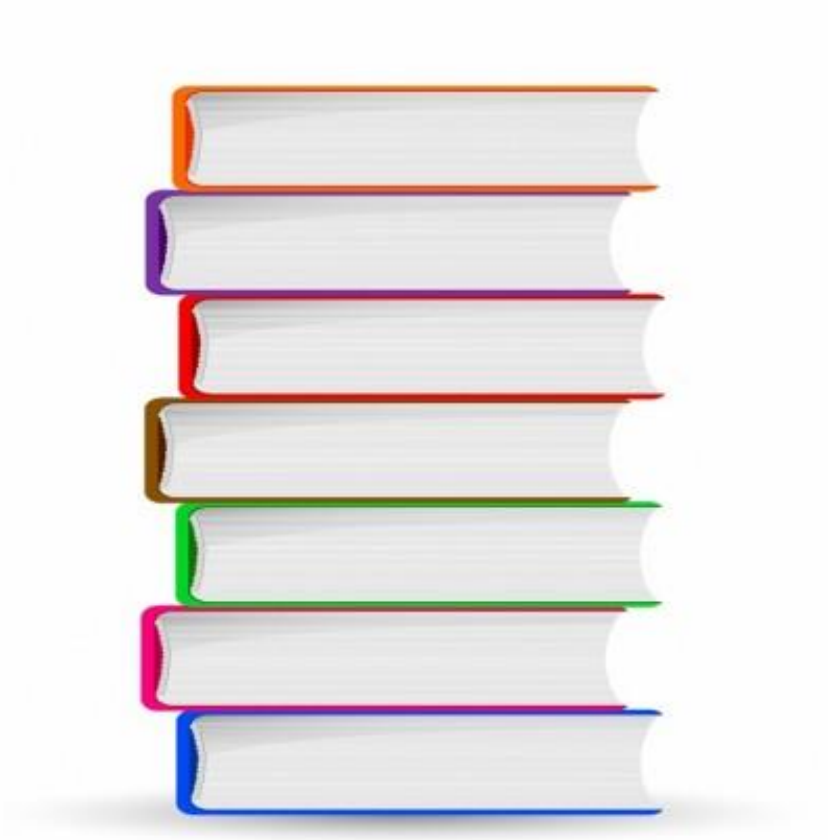
Master/Detail

Progress through pages of data with stack-based navigation

Xamarin University

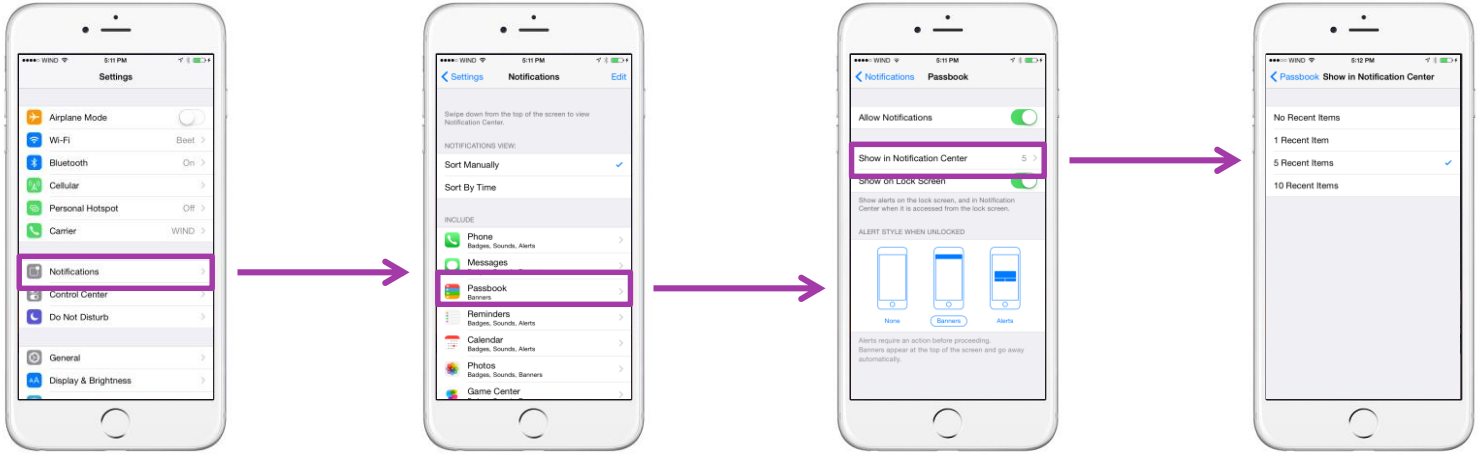
Tasks

1. Create a Navigation Controller programmatically
2. Utilize the designer to create a Navigation Controller
3. Customize the Navigation Controller



Stack navigation

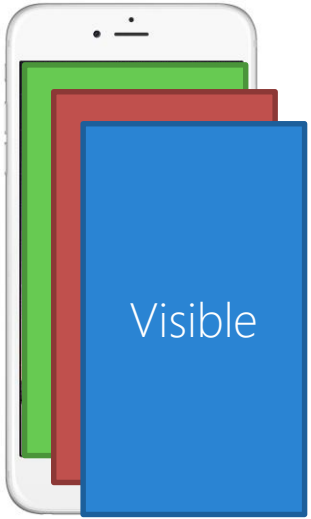
- When we have a hierarchy of data, it's convenient to use stack navigation to browse and interact with the content



Settings Notifications Passbook Notification Center

Stack navigation

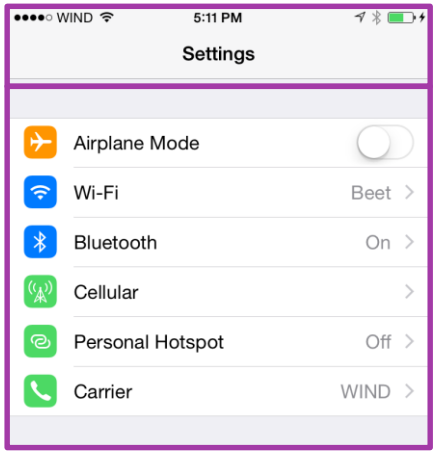
- When a new view controller is *pushed* onto the stack, it becomes visible and hides the previous screen
- Only one view controller is ever visible at a time (the last one added)
- Great for displaying multi-level relationships because it allows "drilling" into details



What is UINavigationController?

- Stack-based navigation is built into iOS through the use of the **`UINavigationController`** class

Displays a Navigation Bar above the currently displayed view controller



Acts as a parent to any number of child view controllers (stored in a stack)

Create a Navigation Controller

- We can create a **UINavigationController** programmatically, most often added as the root view controller for the app

```
public override bool FinishedLaunching(UIApplication application,
                                       NSDictionary launchOptions)
{
    window = new UIWindow (UIScreen.MainScreen.Bounds);
    var navVC = new UINavigationController(new FirstPageVC());

    window.RootViewController = navVC;
    window.MakeKeyAndVisible();
    return true;
}
```

Can pass in the initial view controller to display on the constructor

Forward navigation [programmatically]

- To navigate forward, we add (or "push") a child View Controller onto the Navigation Controller's stack

```
UINavigationController navVC = ...;
...
navVC.PushViewController(newViewController, animated:true);
```

Displays the new View Controller and adds it to the navigation stack, if there was a view controller already shown, then it is hidden and a back button is added to the navigation bar

The NavigationController property

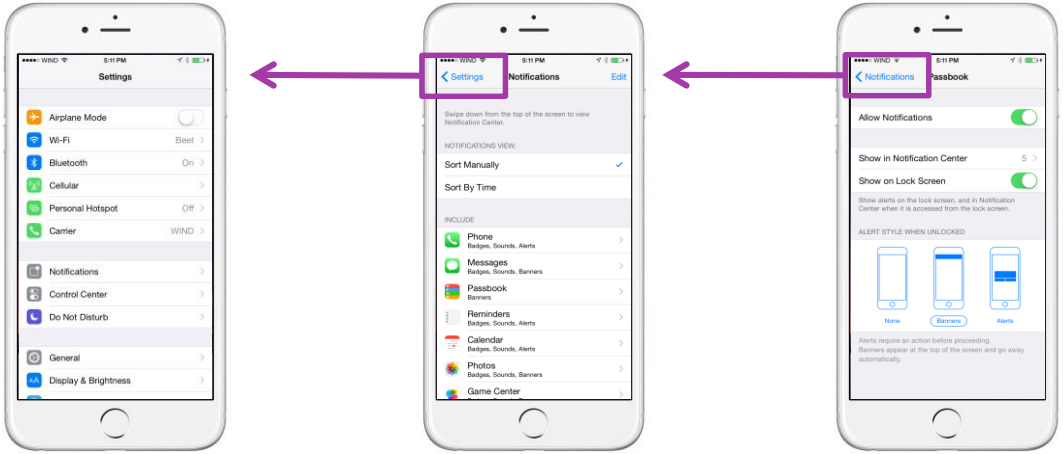
- View Controllers that have been added to the navigation controller can use the **`NavigationController`** property to access it

```
var navCon = myViewController.NavigationController;

navCon.PushViewController(newViewController, animated:true);
```

The **`NavigationController`** property is only valid when this view controller is owned by a navigation controller, otherwise it will be **`null`**!

Back navigation [definition]

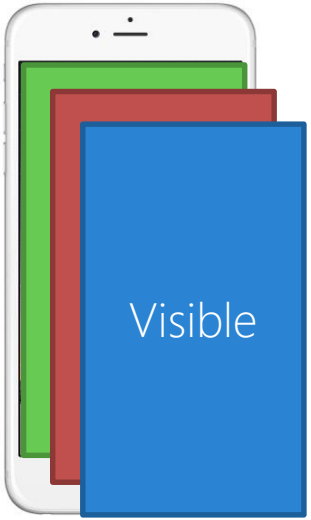
- *Back Navigation* removes the top view controller and navigates back through the stack of child screens



Back navigation

- When a screen is "popped" off the stack, it's removed from the Navigation Controller and the screen below becomes visible

- Must always have at least one view controller on the stack – popping off the last entry will result in an error



Back navigation [programmatically]

- We can navigate back using the **`PopViewController`** method on the Navigation Controller

```
var navVC = new UINavigationController(clockVC);
...
navVC.PopViewController(animated:true);
```

```
navVC.PopToRootViewController(animated:true);
```

Changing the stack

- Navigation controller includes methods to influence the stack directly and properties to interrogate the current state of the navigation stack

```
public class UINavigationController
{
   UIViewController TopViewController { get; }
   UIViewController VisibleViewController { get; }
   UIViewController[] ViewControllers { get; set; }

   UIViewController[] PopToRootViewController(bool animated);
   UIViewController[] PopToViewController(
                        UIViewController viewController, bool animated);
   void SetViewControllers(UIViewController[] controllers, bool animated);
}
```

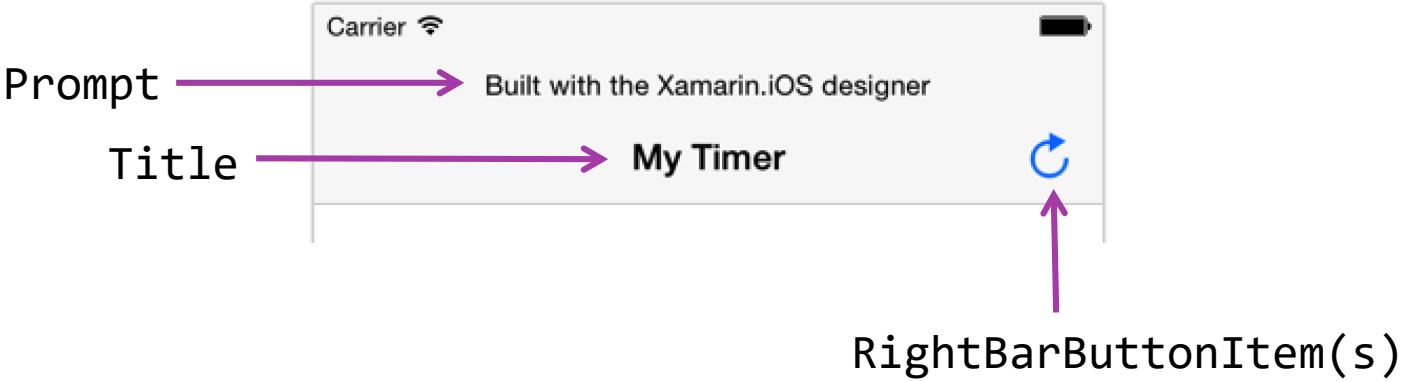
Demonstration

Stack Navigation programmatically

Xamarin University

NavigationItem

- Every View Controller has a **`NavigationItem`** property that can be used to change the behavior and appearance of the Navigation Controller



UIBarButtonItem

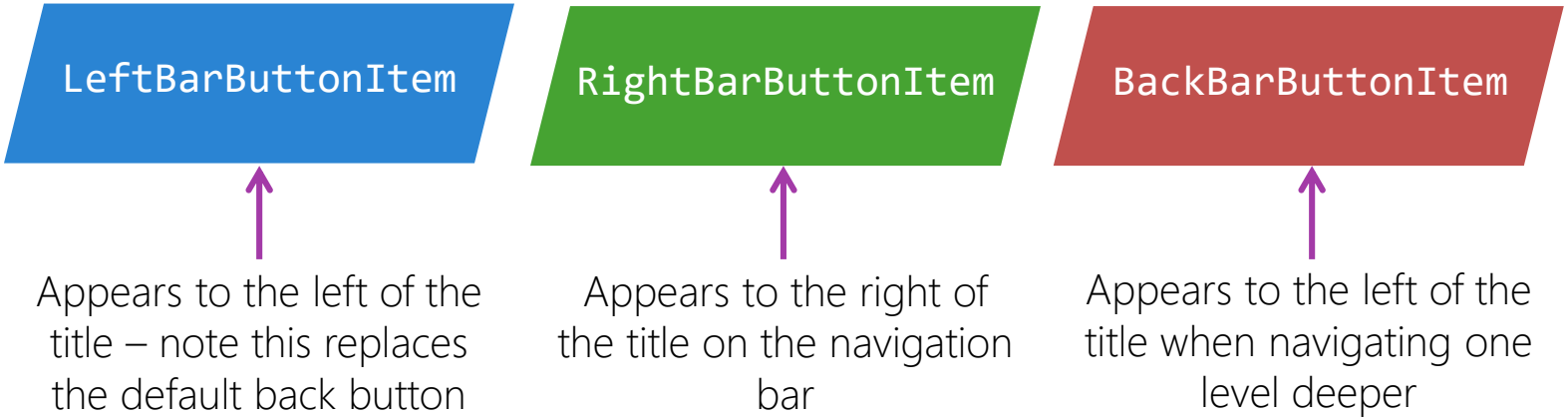
- `UIBarButtonItem` objects can be used to add buttons to the Navigation Bar

`LeftBarButtonItem(s)`

`RightBarButtonItem(s)`

UIBarButtonItems

- The **`UIBarButtonItems`** are available in the **`UINavigationItem`** property



You can add multiple buttons on the left and right sides using the plural forms which take an array of buttons – **`LeftBarButtonItems`** and **`RightBarButtonItems`**

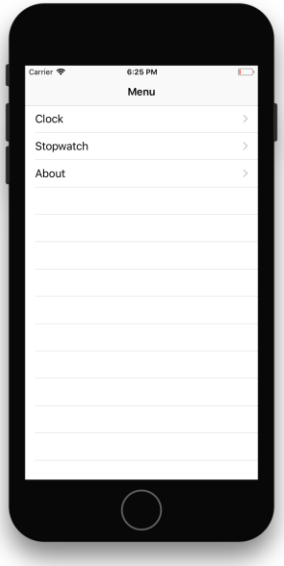
Add BarButtonItems programmatically

- To add bar button items programmatically, set the properties in the currently active **`UIViewController`** using the **`NavigationItem`** property

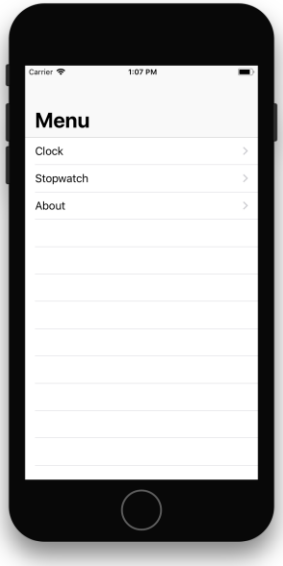
```
this.NavigationItem.LeftBarButtonItem = new UIBarButtonItem(...);

this.NavigationItem.RightBarButtonItem = new UIBarButtonItem(...);

this.NavigationItem.BackBarButtonItem = new UIBarButtonItem(...);
```

Navigation bar title size

- iOS supports two sizes for the page title displayed in the navigation bar



Normal



Large

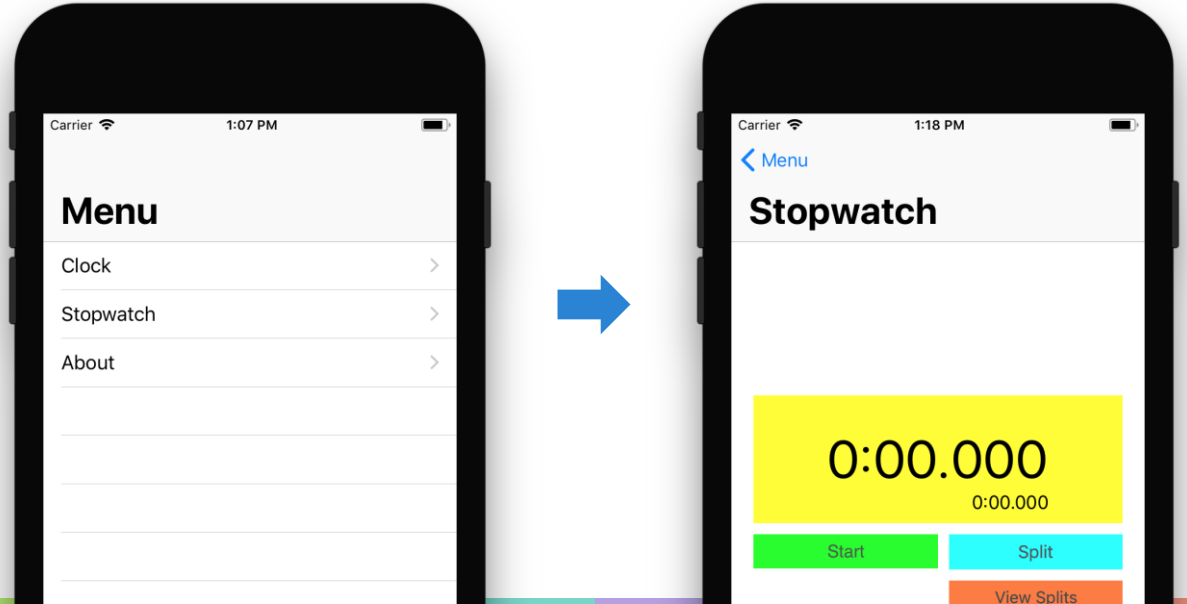
Navigation bar large titles

- **UINavigationBar** can display large titles by setting the **PrefersLargeTitles** property to true

```
public override void ViewDidLoad()
{
   NavigationController.NavigationBar.PrefersLargeTitles = true;
}
```

Title size inheritance

- By default, setting **PrefersLargeTitles** to true will cause child pages to have large titles as well



Participating in large titles

- **NavigationItem** has a property called **LargeTitleDisplayMode** to control whether a view controller wants to participate in having large titles

```
public override void ViewDidLoad()
{
    NavigationItem.LargeTitleDisplayMode = UINavigationItemLargeTitleDisplayMode.
}
```

Always
Automatic
Never

The default value is Automatic

Hiding the Navigation Bar

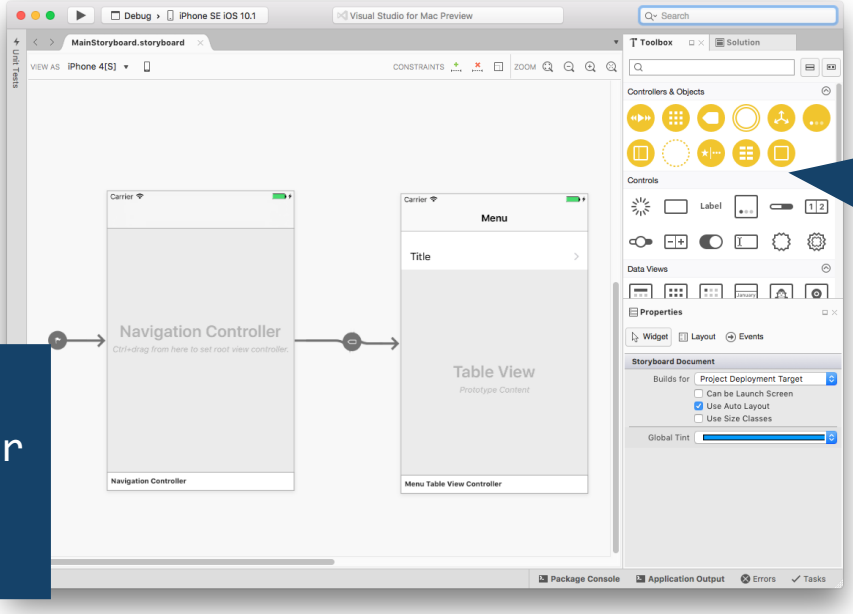
- The visibility of the navigation bar can be changed by setting properties on the **`UINavigationController`**

```
P BarHideOnSwipeGestureRecognizer
P BarHideOnTapGestureRecognizer
M ChildViewControllerForStatusBarHidden
P HidesBarsOnSwipe
P HidesBarsOnTap
P HidesBarsWhenKeyboardAppears
P HidesBarsWhenVerticallyCompact
P HidesBottomBarWhenPushed
P NavigationBarHidden
```

These properties are accessed directly from an instance of a **`UINavigationController`**

UINavigationController in the Designer

- We can add a **`UINavigationController`** to a Storyboard using the iOS Designer

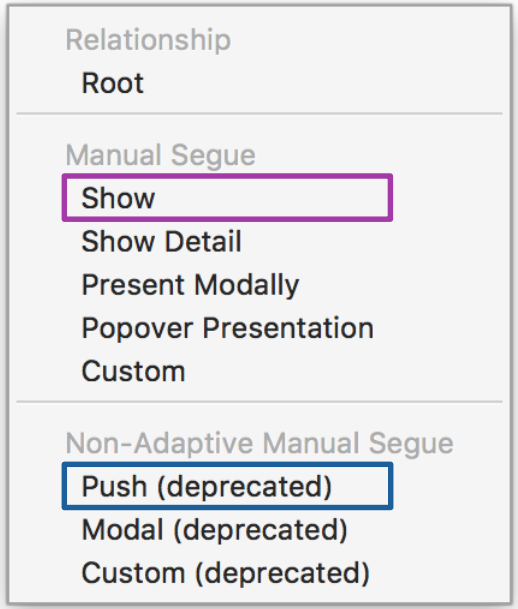


Use the Toolbox and search for Navigation Controller

Design provides a **`UINavigationController`** and one child **`UIViewController`**

Navigation using Segues

- To add view controllers to the navigation stack using the designer - use the Show Segue



Prepare for Segue

- To interact with a View Controller before it's displayed via a Segue, override the **PrepareForSegue** method on the source View Controller

```
public override void PrepareForSegue(
    UIStoryboardSegue segue, NSObject sender)
{
    base.PrepareForSegue (segue, sender);

    var aboutVC = segue.DestinationViewController as
        AboutViewController;
    ...
}
```

Must cast the destination View Controller to access custom properties and methods

Instantiating a View Controller

- To instantiate a View Controller defined in a Storyboard programmatically, use the **`InstantiateViewController`** method

```
UIViewController controller = ...;
var sb = controller.Storyboard;

var newVC = sb.InstantiateViewController ("myViewController");

controller.PushViewController (newVC, true);
```

Can then push the new controller onto the navigation stack

The parameter must be a valid storyboard identifier

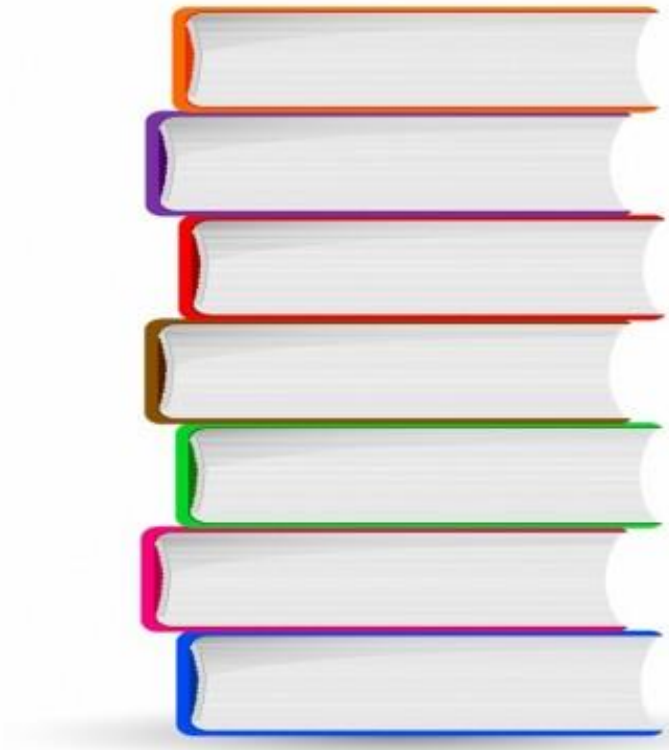
Individual Exercise

Add a UINavigationController to a storyboard

Xamarin University

Summary

1. Create a Navigation Controller programmatically
2. Utilize the designer to create a Navigation Controller
3. Customize the Navigation Controller



Show different views of related data with tab navigation

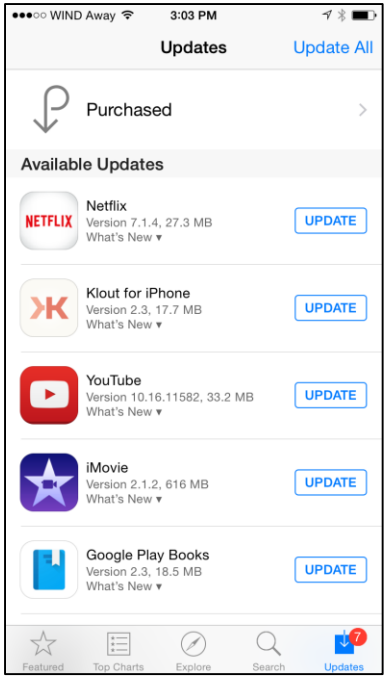
Xamarin University

Tasks

1. Create a Tab Bar Controller
2. Populate a Tab Bar Controller
3. Customize the Tab Bar Controller

Tab navigation

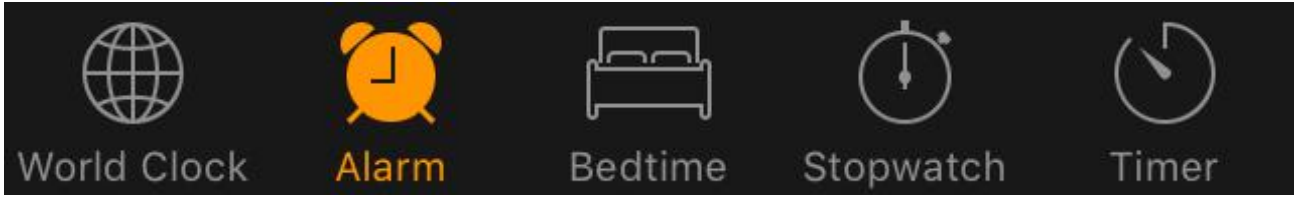
- Tab navigation allows users to quickly switch between view controllers by selecting tabs displayed at the bottom of the screen
  - Ideal for 3-5 screens of equal importance



The active page's tab is highlighted

What is a UITabBarController?

- iOS implements tab navigation with the **`UITabBarController`**



Displays tabs at the bottom of the screen,
each tab can show an icon and a label

A **`UITabBarController`** can hold a **`UINavigationController`** within a tab

UITabBarItem

- A **`UITabBarItem`** is an object used to describe the appearance of a single tab within a Tab Bar Controller

```
var tbi = new UITabBarItem("Clock", UIImage.FromBundle("clock.png"), 0);
```

`UITabBarItem` constructor takes a title, an image, and an integer "tag" which can be used to identify the item later

Using the UITabBarItem

- Every View Controller has a **`TabBarItem`** property which can be set to an instance of a **`UITabBarItem`**

```
var tabViewController = new ClockViewController();
var tbi = new UITabBarItem("Clock", UIImage.FromBundle("clock.png"), 0);
tabViewController.TabBarItem = tbi;
```

Setting the **`TabBarItem`** property will determine the title and image shown on a tab for this view controller when it's added to a tab bar controller

UITabBarController programmatically

- Add children to tab controller using the **`ViewControllers`** property

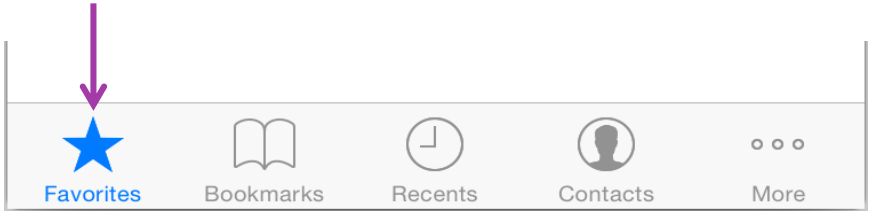
```
public class MyTabBarController : UITabBarController
{
   public override void ViewDidLoad()
   {
      base.ViewDidLoad();

      var vc1 = new ClockViewController {
         TabBarItem = new UITabBarItem("Clock",
                              UIImage.FromBundle("clock"), 0);
      };
      ...
      this.ViewControllers = new UIViewController[] { vc1, vc2, vc3 };
   }
}
```

The **`ViewControllers`** property is set to an array of **`UIViewControllers`**

TabBar images

- Images can be set on the tabs; resource images or system image can be displayed

Image base size should be at least 25x25 (32x32 is ideal)



Prefer monochromatic, transparent images for template (stencil) filtering

Remember to include **@2x** (64x64) and **@3x** (96x96) images for high-resolution devices

UITabBarSystemItem

- **UITabBarSystemItem** provides a small selection of built-in images that can be used to decorate the tabs

Each item has a set title that cannot be changed

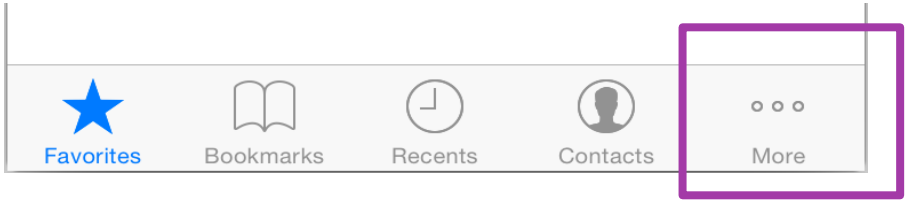
```
var tab1 = new FavoritesViewController ();

tab1.TabBarItem = new UITabBarItem (UITabBarSystemItem.Favorites, 0);
```

Favorites

UITabBarSystemItems can also be set from the property panel in the designer

Overflowing the UITabBarController

- The **`UITabBarController`** can show up to **5 tabs** on the iPhone and **8 tabs** on the iPad or iPhone 6+; if more tabs are added then the system creates a "more" tab and displays the remainder in a system-provided Table View



Overflow (more) tab

Detecting tab selection

- To respond to selection events on the **`UITabBarController`**, subscribe to the **`ViewControllerSelected`** event handler

```
public class ClockTabBarController : UITabBarController
{
   public override void ViewDidLoad()
   {
      ...
      this.ViewControllerSelected += TabSelected;
   }
   ...
}
```

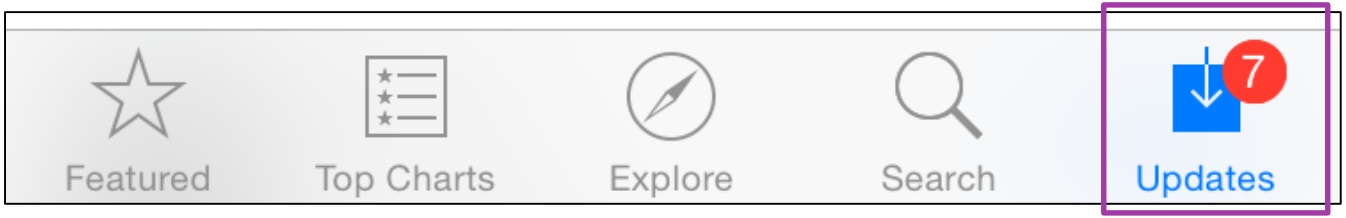
TabBar selected item

- To determine which tab is selected, use **`TabBar.SelectedItem`** property

```
public class ClockTabBarController : UITabBarController
{
    ...
    void TabSelected (object sender, UITabBarSelectionEventArgs e)
    {
        var alert = new UIAlertView("Tab tapped",
            this.TabBar.SelectedItem.Title, null, "OK", null);
        alert.Show();
    }
    ...
}
```

Tab Badges

- A *Badge* can be added to a tab to display a small amount of text by setting the **`BadgeValue`** property on a **`UITabBarItem`**



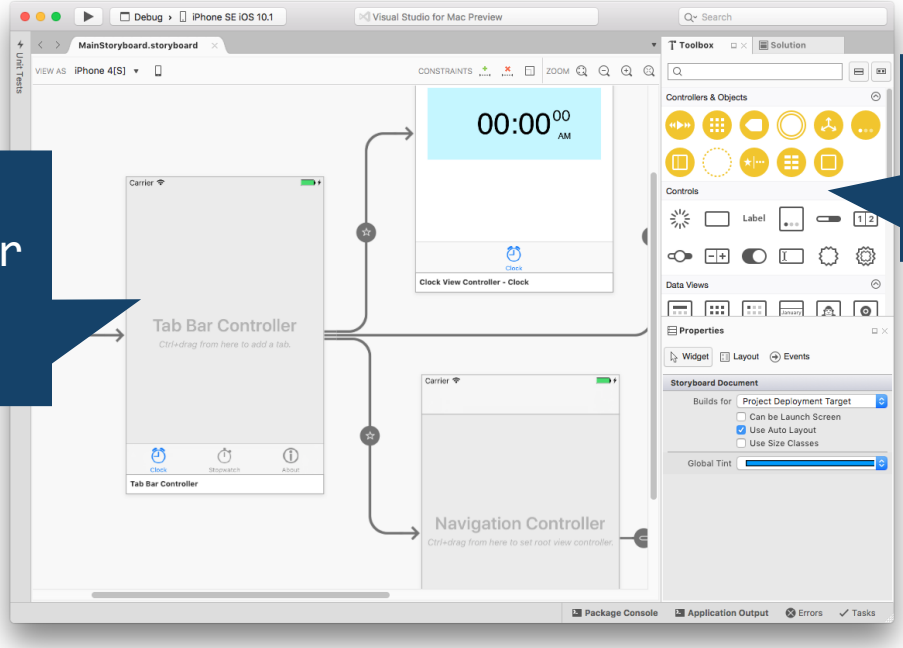

```
updatesVC.TabBarItem.BadgeValue = "7";
```

```
updatesVC.TabBarItem.BadgeValue = null;
```

Add a Tab Bar Controller to a Storyboard

- The Xamarin iOS designer can be used to add a **`UITabBarController`** to a Storyboard

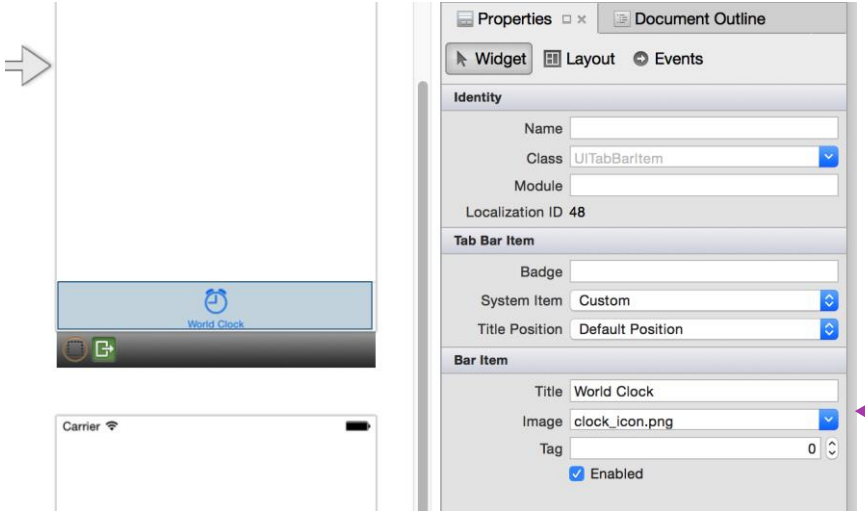
Provides a **`UITabBarController`** and two child **`UIViewControllers`**



Use the Toolbox and search for TabBar Controller

Customizing the Tabs from the Designer

- The designer will show additional UI and properties when a child view controller is connected to a **`UITabBarController`** via a Segue



Set the title and tab bar image

Individual Exercise

Add a UITabBarController to a Storyboard

Xamarin University

Summary

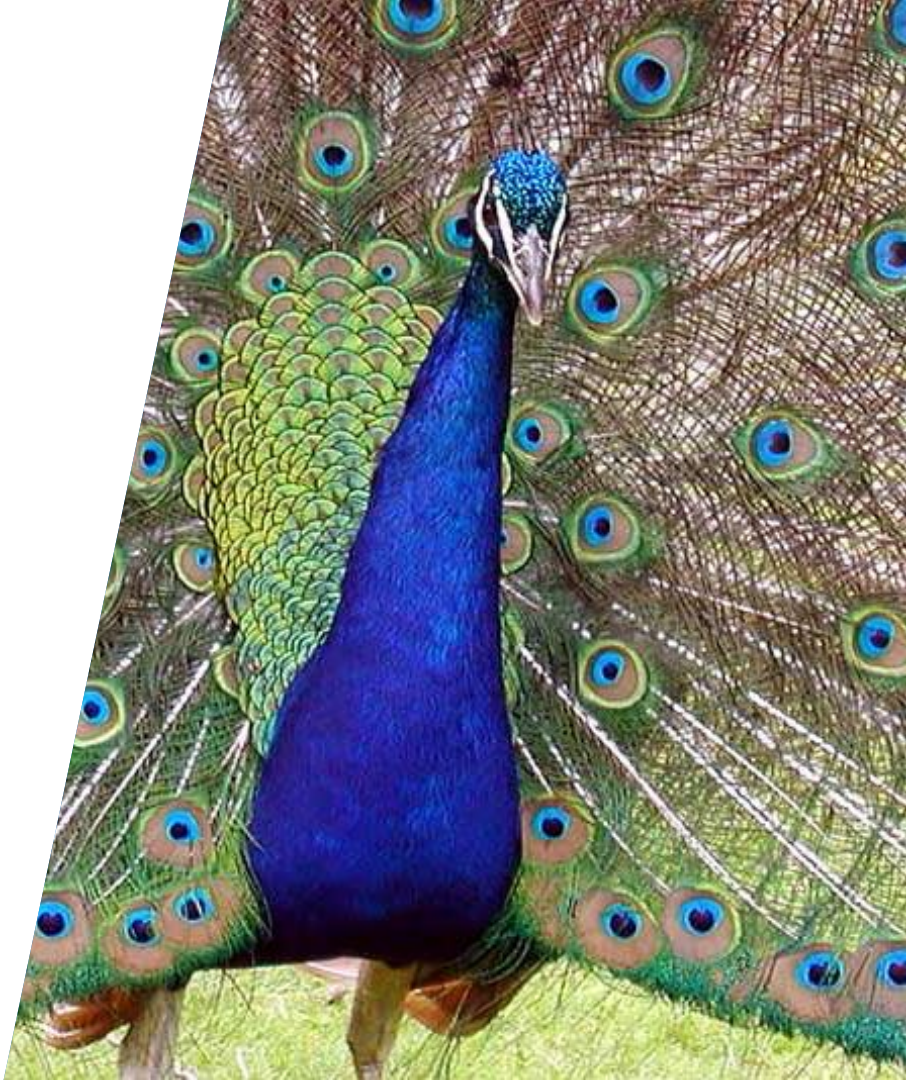
1. Create a Tab Bar Controller
2. Populate a Tab Bar Controller
3. Customize the Tab Bar Controller

Display hierarchical relationships with master/detail navigation

Xamarin University

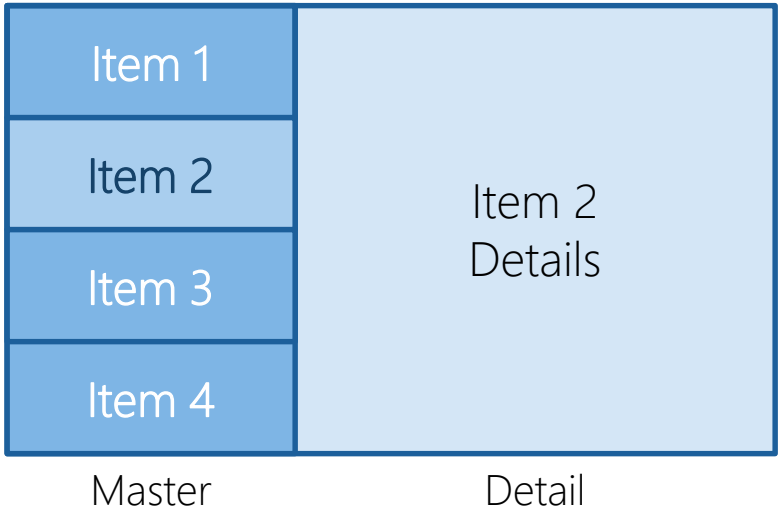
Tasks

1. Create a Split View Controller
2. Use a Split View Controller programmatically
3. Use the iOS Designer to define a Split View Controller



Master/Detail navigation

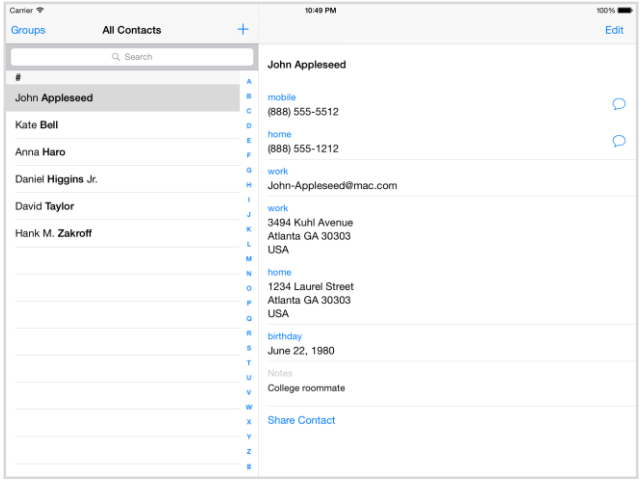
- A Master/Detail navigation pattern displays a “Master” list used for primary navigation along side a second visual area displaying the “Details” for the currently selected item



What is the UISplitViewController?

- The **`UISplitViewController`** class manages the display of two side-by-side view controllers

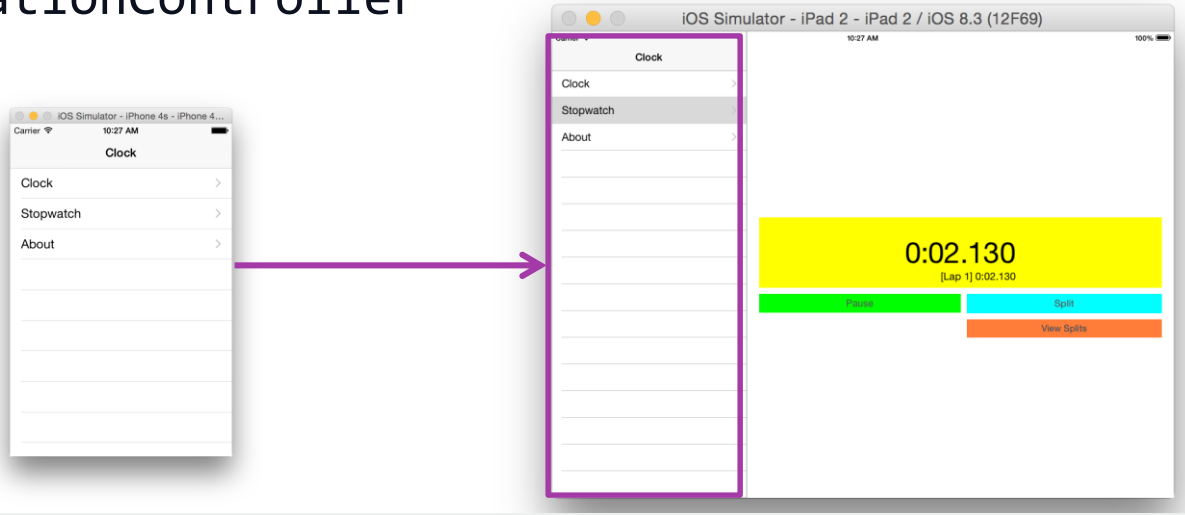
Left displays a list of items for navigation (Master)



Right displays details about the selected item (Detail)

Master/Detail Responsive design

- When the Split View Controller is used on smaller displays (iPhone), it "collapses" so only one View Controller is shown, effectively mimicking a **`UINavigationController`**



Navigation on tablets is limited to two levels. Best practice is to limit to two levels for consistent behavior on phone and tablet.

Creating a SplitView Controller

- When creating a **`UISplitViewController`** programmatically, the master/detail views are assigned to the **`ViewControllers`** property

```
public class EventSplitViewController : UISplitViewController
{
    MasterViewController masterView;
    DetailViewController detailView;

    public EventSplitViewController() : base()
    {
        masterView = new MasterViewController();
        detailView = new DetailViewController();

        ViewControllers = new UIViewController[] { masterView, detailView };
    }
}
```

Navigating programmatically

- There is an adaptive method **`ShowViewController`**, defined on the **`UIViewController`** which is used for forward navigation

```
public void ShowViewController(
     UIViewController controller, NSObject sender)
```

Sets the master view, or current navigation view, or modal view

Navigating programmatically

- The **`ShowDetailViewController`** method is used to update the detail view in a Split View Controller

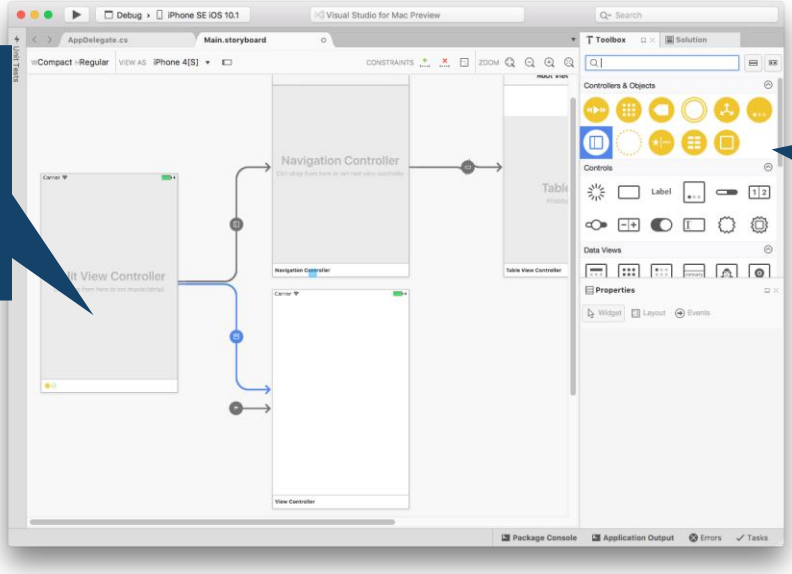
```
public void ShowDetailViewController(
     UIViewController controller, NSObject sender)
```

Replaces the detail view (right side of a split view)

Add a Split View Controller [Designer]

- The Xamarin iOS designer can be used to add a **`UISplitViewController`** to a Storyboard

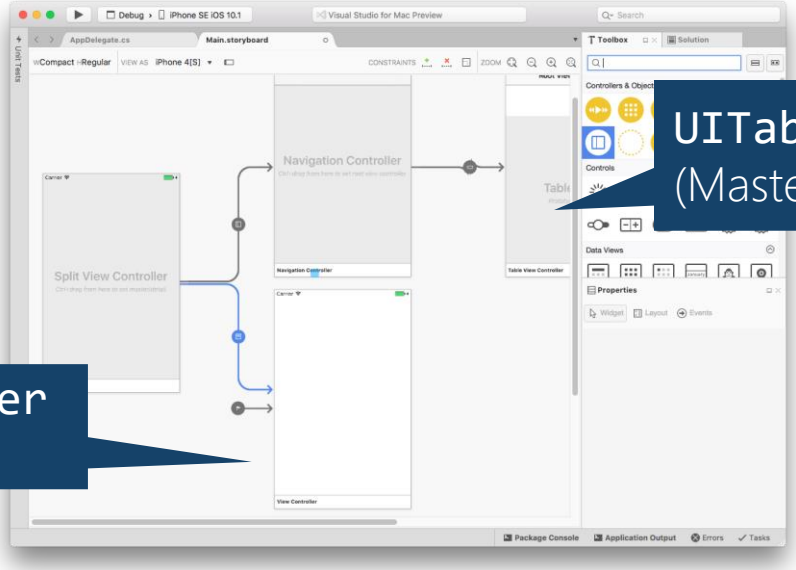
Provides a **`UISplitViewController`** and three child view controllers



Use the Toolbox and search for Split View Controller

Master/Detail View Controllers

- The Xamarin iOS designer provides a **`UITableViewController`** within a **`UINavigationController`** for the Master UI, and a simple **`UIViewController`** for the details UI



`UITableViewController` (Master)

`UIViewController` (Detail)

Flash Quiz

Xamarin University

Flash Quiz

① Which method would you use to replace the Master view for a `UISplitViewController`?

a) PushViewController

b) PresentModalViewController

c) ShowViewController

Flash Quiz

① Which method would you use to replace the Master view for a `UISplitViewController`?

a) PushViewController

b) PresentModalViewController

c) ShowViewController

Flash Quiz

② The `UISplitViewController` will show both the master and detail views when displayed on an iPad Mini

a) True

b) False

Flash Quiz

② The `UISplitViewController` will show both the master and detail views when displayed on an iPad Mini

a) True

b) False

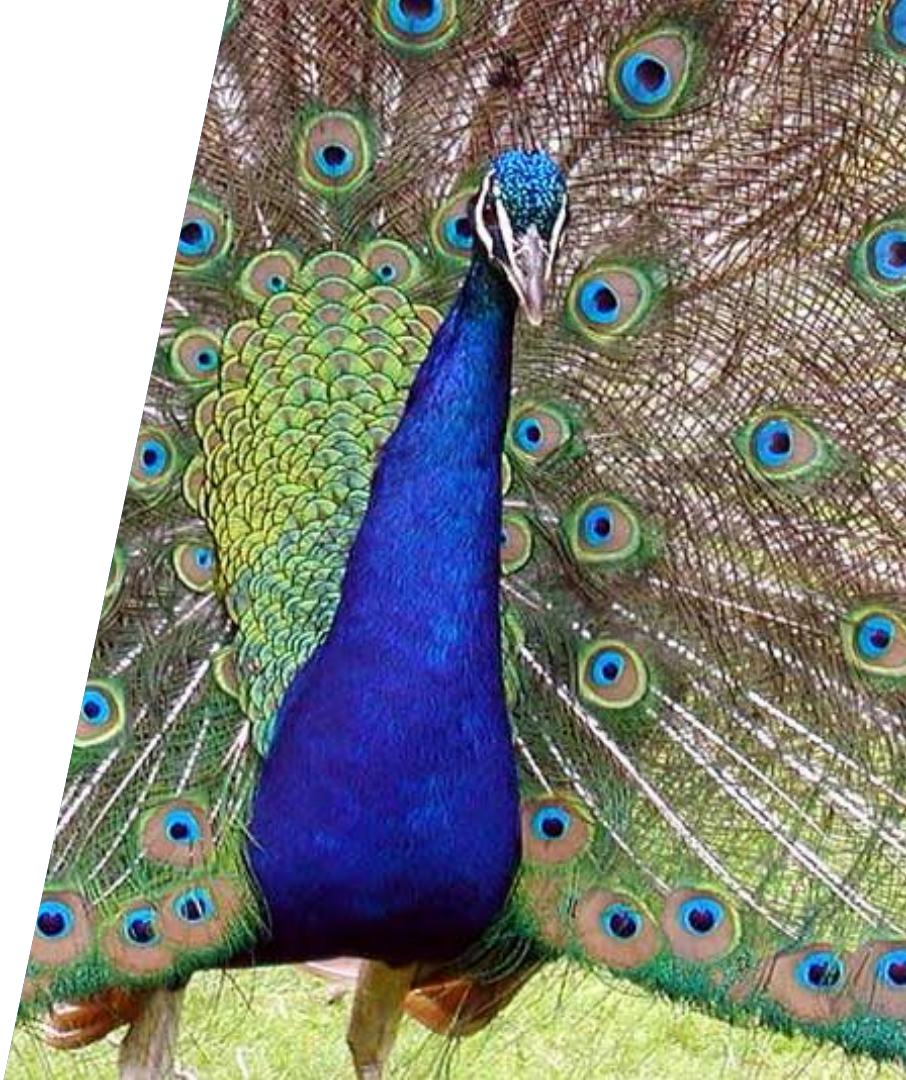
Individual Exercise

Add a UISplitViewController to a Storyboard

Xamarin University

Summary

1. Create a Split View Controller
2. Use a Split View Controller programmatically
3. Use the iOS Designer to define a Split View Controller



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft