Example Application Flow

Tab Bar Controller → Navigation Controller → View Controller

•••
Example Flow – Terms

Top Level

Root

Interior / List

Top / Detail

TableView Controller

TableView Controller

TableView Controller

Navigation Controller
Typical Navigation

• “Root” view controller
  - Often `UITabBarController` or `UINavigationController`
  - Created in Application Delegate

• Navigation controller interior items
  - Respond to events and create new view controllers
  - Push onto navigation stack, or display modally
Tab As Root View

• Setup in Application Delegate

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    // Create top level tab bar controller
    tabBarController = [[UITabBarController alloc] init];

    // Create each individual tab item’s controller
    // Initialized with a root controller
    vctl1 = [[[SomeTableViewController alloc] init ...] autorelease];
    vctl2 = [[[UINavigationController alloc] init ...] autorelease];

    // Give the individual tab item’s to the tab controller
    NSArray *subitems = [NSArray arrayWithObjects: vctl1, vctl2, nil];
    [tabBarController setViewControllers: subitems];

    // Add the tab controller’s view
    // This will cause -loadView to execute, or the XIB to load
    [window addSubview:tabBarController.view];

    // Put it all on screen
    [window makeKeyAndVisible];
}

Monday, October 4, 2010
Descending Navigation Hierarchy

• Respond to an action, or table selection

- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    // Find info to display in the detail view
    // We will pass this data to the detail view controller
    id info = [self infoForRowAtIndexPath:indexPath];
    if (info) {
        // Create a DetailViewController
        DetailViewController *dvc = [[DetailViewController alloc] init];

        // Pass it the info it needs
        [dvc setDetailInfo:info];

        // Animate it in the current navigation stack
        [[self navigationController] pushViewController:dvc animated:YES];

        // The navigation controller owns the DetailViewController
        // as a result of it being on the current stack
        [dvc release];
    }
}
Navigating Back

- Back button – leave it up to the user
- Programmatically

```
[navController popViewControllerAnimated:YES]
```
Data Flow

- Determine what data needs to be communicated
- Provide mechanisms for passing data to sub controllers
  - Setter methods, properties
  - Parameters to your own designated initializer
- For communicating back up the hierarchy, use loose coupling
  - Your own notifications
  - You own delegation or data source protocol
Integrating View Controllers

- `UIViewController` provides access to container adornments
Fitting In With TabBarController

• Each Tab has a corresponding view controller

• Can configure tab item & access containing controller

@interface UIViewController (UITabBarControllerItem)

// Returns the nearest containing tab bar controller
@property(nonatomic,readonly,retain)UITabBarController *tabBarController;

// Default is an item that displays the view controller’s title.
// Created lazily, so don’t use this if not in a tab bar
@property(nonatomic,retain)UITabBarItem *tabBarItem;
@end
Fitting In With NavigationController

• API to configure containing controller’s navigation bar

```swift
@interface UIViewController (UINavigationControllerItem)

// Default is an item that uses the view controller’s title (title, back button)
// Created lazily, so don’t use this if not in a navigation controller
@property(nonatomic,readonly,retain) UINavigationItem *navigationItem;

// Default is NO. If YES, pushing this view controller hides any bottom bar
@property(nonatomic) BOOL hidesBottomBarWhenPushed;

// Returns the nav controller containing this view controller.
@property(nonatomic,readonly,retain) UINavigationController *navigationController;
@end
```

• `navigationItem` – default settings usually fine

• `hidesBottomBarWhenPushed` – hides toolbars, button bars
Hides Bottom Bar

Example: Toolbar
Fitting In With Navigation Controllers

- More `UIViewController` categories focused on toolbars

```objc
@interface UIViewController (UINavigationControllerContextualToolbarItems)
// The list of UIBarButtonItem to display in a navigation toolbar
// Configure before display or any time after...
@property (nonatomic, retain) NSArray *toolbarItems;
@end
```
Toolbar Example

• Per-View Controller toolbar items
Modality
Modal

• Modal interfaces restrict user interaction to a single window
  ▪ iPhone modal controllers cover screen, disabling interaction with other view controllers

• Used to temporarily interrupt application flow
  ▪ Present / gather user information before proceeding
Presenting Modal Controllers
Presenting Modal Controllers

AddEventViewController *modalVC = ...;
[calVC presentModalViewController:modalVC animated:YES];
Presenting Modal Controllers

AddEventViewController *modalVC = ...;
[calVC presentModalViewController:modalVC animated:YES];
Presenting Modal Controllers

[calVC dismissModalViewControllerAnimated:YES];
Presenting Modal Controllers

```objective-c
    [-calVC dismissModalViewControllerAnimated:YES];
```
Presenting Modal Controllers

• Presenting
  ▪ Attach to the topmost view controller to ensure screen is covered

• Dismissing
  ▪ Preferred approach is to let parent view controller dismiss
    ▪ Parent view acts as delegate for modal view controller
    ▪ Usually done in response to a “done” callback
Modal Transition Styles

viewController.modalTransitionStyle = UILModalTransitionStyleVertical;
viewController.modalTransitionStyle = UIModalTransitionStyleVertical;
viewController.modalTransitionStyle = UIModalTransitionStyleCrossFlipHorizontal;
viewController.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
viewController.modalTransitionStyle = UIModalTransitionStyleCrossDissolve;
Prompt

- Modal views often display explanatory text
- Prompt attribute available in UINavigationItem

UINavigationItem *item = [self navigationItem];
[item setPrompt: @"Set the details for this event."];
Modal Controller Chain

- UIViewController modal view controller hierarchy

```
// Reference to the modal view controller starting point
@property(nonatomic,readonly) UIViewController *modalViewController;

// If inside a navigation or tab controller returns that view controller
// If a modal view controller, returns view controller it is attached to...
@property(nonatomic,readonly) UIViewController *parentViewController;
```

- Can be starting point for a chain of view controllers
  - Modal controllers can even present other modal view controllers

- Many system features exposed as view controllers
  - UIImagePickerController
  - ABPeoplePickerNavigationController
Rotation
Rotation

• Checklist for Custom View Controllers

  ▪ Return YES from `-shouldAutorotateToInterfaceOrientation`:

  ▪ Setup `autoresizing` masks to handle supported orientations

  ▪ Implement orientation change steps

    ▪ can use either one- or two-step process
One Step Rotation

- Your view controller receives
  
  - (void)willRotateToInterfaceOrientation:...

  ▪ Hide any views necessary
  ▪ Make changes to view layout before rotation

- Then you receive
  
  - (void)didRotateFromInterfaceOrientation:...
Two Step Rotation

- Additional override points

- First Half

  -(void)willAnimateFirstHalfOfRotationToInterfaceOrientation:
  -(void)didAnimateFirstHalfOfRotationToInterfaceOrientation:

- Second Half

  -(void)willAnimateSecondHalfOfRotationToInterfaceOrientation:

- Use

  - Typically used to hide a portrait version of some UI

  - E.g. Video Player controls fade out, then back in
Orientation Based Interface

• Example
  ▪ Calculator application
  ▪ Portrait – simple calculator
  ▪ Landscape – scientific calculator

• Easiest Solution
  ▪ Use multiple view controllers
  ▪ One view controller for each orientation
Orientation Based Interface

- **MyPortraitViewController**
  - Declare support for only portrait
  - Using `UIDevice`, watch for orientation changes *notifications*
    - Normally you implement overrides, but you’ve disabled the auto rotation

- **MyLandscapeViewController**
  - Managed by the portrait view controller

- **Use modal presentation!**
  - Present landscape view modally when necessary
Orientation Based Interface

- How To...

```objective-c
- (void)orientationChanged:(NSNotification *)notification
{
    UIDeviceOrientation deviceOrientation = [UIDevice currentDevice].orientation;

    if (UIDeviceOrientationIsLandscape(deviceOrientation) &&
        !isShowingLandscapeView)
    {
        [self presentModalViewController:self.landscapeViewController
            animated:YES];
        isShowingLandscapeView = YES;
    }

    else if (deviceOrientation == UIDeviceOrientationPortrait &&
        isShowingLandscapeView)
    {
        [self dismissModalViewControllerAnimated:YES];
        isShowingLandscapeView = NO;
    }
}
```
Orientation Based Interface

• How To...

- (void)orientationChanged:(NSNotification *)notification
{
    UIDeviceOrientation deviceOrientation = [[UIDevice currentDevice] orientation];
    if (UIDeviceOrientationIsLandscape(deviceOrientation) &&
        !isShowingLandscapeView)
    {
        [self presentModalViewController:self.landscapeViewController
                                  animated:YES];
        isShowingLandscapeView = YES;
    }
    else if (deviceOrientation == UIDeviceOrientationPortrait &&
             isShowingLandscapeView)
    {
        [self dismissModalViewController:animated:YES];
        isShowingLandscapeView = NO;
    }
}
Editing Support
Editing

- **UIViewController** provides hooks to deal with editing

- Access to the configured “edit” bar button

  ```
  -(UIBarButtonItemAt)editButtonItem;
  ```

- Override point, called by standard “edit” bar button

  ```
  -(void)setEditing:(BOOL)editing animated:(BOOL)animated;
  ```

- Standard behavior

  - **UIViewController** toggles button UI
  
  - **UITableViewViewController** forwards edit action to its table
Tab Bar Customization

- **UITabBarController** provides API to support user reordering

```objective-c
// If not empty, the “More” screen will show an “edit” button
@property(nonatomic, copy) NSArray *customizableViewControllers;

@protocol UITabBarControllerDelegate <NSObject>
// viewControllers represents the new display order chosen by the user
- (void)tabBarController:(UITabBarController *)tabBarController
didEndCustomizingViewControllers:(NSArray *)viewControllers
  changed:(BOOL)changed;
@end
```
Review
Review

• Custom View Controller Checklist
  ▪ Create UI in `-loadViews`, or in IB (and implement `-nibName`)
  ▪ Define methods for passing data
  ▪ Define delegate methods / notifications for communicating changes
  ▪ Memory efficiency (`-viewDidUnload`, `-didReceiveMemoryWarning`)
  ▪ Rotation Support
  ▪ Define actions, implement to push new controllers on the stack
  ▪ Configure UI - nav item, tab bar item, toolbar items
Review

• Application Flow
  - Create top level controller
  - For Tab Bar Controller
    - Create controllers for each tab, and call `setViewControllers`: 
  - For Navigation Controller
    - Create and connect `root controller`, call `setRootController`: 

• Saving application state
  - Need to remember stack of nav items, etc...
  - Will discuss in lectures dealing with “data persistence”
Reading

- “View Controller Programming Guide For iPhone OS”
  - ViewControllerPGforiPhoneOS.pdf
  - Read “Custom View Controllers” - p.21 - 52
  - Read “Combined View Controllers” - p.97 - 103

- iPhone OS Reference Library
  - UINavigationController Class Reference
    - Supporting Classes UINavigationItem, UIBarButtonItem