Technical Presentation

---- Overview of programming structure of an IOS application

Design Team 10:
Ryan Bliton
Feifei Li
Mark Mudri
Kaichang Wang
Charlton Washington
Kyler Wilkins
Outline

➔ What is Xcode?
➔ Xcode Structure
➔ View Controllers
➔ Header files
➔ Source(.m) files
➔ Conclusion
What is Xcode?

➔ An integrated development environment (IDE)
➔ Used to develop OS and iOS applications
➔ Only available on Mac Computers
➔ First released in 2003, Now on version 5.0
➔ Contains a suite of software development tools developed by Apple
  ◆ graphical user interface
  ◆ source code editor
  ◆ code compiler
  ◆ debugger
  ◆ simulator
Xcode Structure

➔ Creating a New Project
  ◆ Project Templates
➔ Hierarchy Flowchart
➔ Storyboards
➔ Utilities
➔ Objects
Choose a template for your new project

- iOS
  - Application
    - Framework & Library
    - Other
  - Master-Detail Application
  - OpenGL Game
  - Page-Based Application
  - Single View Application
  - Tabbed Application
  - Utility Application
  - Empty Application
  - SpriteKit Game

Empty Application

This template provides a starting point for any application. It provides just an application delegate and a window.

Cancel  Previous  Next
Project Templates

➔ Single View

➔ Tabbed

➔ OpenGL Game
Project Templates

➔ Page View

➔ Utility

➔ Master Detail
```
#import "XYZAppDelegate.h"

@implementation XYZAppDelegate

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [(UIWindow alloc) initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch.
    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application
{
    // Sent when the application is about to move from active to inactive state. This can occur for certain types
    of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits
    the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers,
```
Utilities

➔ Inspector Window
◆ Help
◆ Identity
◆ Size
◆ Connections

➔ Objects
Objects

Button

UIButton

Implements a button that intercepts touch events and sends an action message to a target object when it's tapped. You can set the title, image, and other appearance properties of a button. In addition, you can specify a different appearance for each button state.

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.
Simulator

➔ Multiple Simulation Devices Available
➔ Allows Touch, Swipe, Keyboard
View Controllers

→ Manages the view objects and provides them with behavior

→ Play many roles:
  ◆ coordinate the flow of information
  ◆ manage the life cycle
  ◆ handle orientation changes

→ ViewController.h & ViewController.m
➔ Navigation controller  ➔ Split view controller
➔ Tab bar controller  ➔ Popover controller
➔ Page view controller  ➔ Combined

developer.apple.com
Header (.h) files

- Directive in Objective C - “import”
- Used for class declarations
- Helpful for using different parts together

```objective-c
#import <Foundation/NSObject.h> // EXAMPLE - FRACTION.h
@interface Fraction : NSObject {
    int numerator;
    int denominator;
}
- (void) setNumerator: (int) n;
- (void) setDenominator: (int) d;
- (int) numerator;
- (int) denominator;
@end
```
Source (.m) files

- Controls implementation of an object when user action is detected
  - Example: Button pressed, changes text in a label

- Sample code:
  ```c
  - (IBAction)Button:(id)sender
  {
    label1.text = [NSString stringWithFormat:@"New Text"];
  }
  ```
Fraction files

Fraction.h

#import <Foundation/NSObject.h>
@interface Fraction: NSObject {
    int numerator;
    int denominator;
}
-(void) print;
-(void) setNumerator: (int) n;
-(void) setDenominator: (int) d;
-(int) numerator;
-(int) denominator;
@end
Important Xcode Techniques: Connecting Interface Pages

1. Select the button, the **Control+Click** and drag from the button to the next screen.

2. Select **push** if connecting to a different navigation controller, select **modal** if not.

3. Click the segue (link) and give it a name to allow programmatic control.
Important Xcode Techniques:

Connecting Screens to Code

1. Create a new **Cocoa Touch Objective-C Class** file. Name it to match the screen.

2. Make this match the type of view controller.

3. Select the screen and under the **Identity Inspector** tab set **Custom Class** to the new class file.
Important Xcode Techniques:

Connecting Interface Objects to Code

1. Select the button, **Control+Click** and drag it into the new .h file below ‘@interface’.

2. Add a name, set **Connection** to ‘Action’ and switch **Type** to ‘UIButton’.

3. Click Connect and a control definition will be added to the code. Now the button can be controlled by “including” this .h file.
3 minute demo video inserted here.
Question?