**Swift Class**

**Broadcastr**

If you don't already have these programs installed, download "Sketch", "Principal for Mac" and "Postman" You will need to select 30-day trials for Sketch and Principal.

Today's Objective: implement a UITableView

Step 1: Create a new class called "MessageTableView" in the "Custom Controls" folder. Make this a subclass of "UITableView"

Step 2: Open the file "UIController.swift" and create a class called "UIController" which is a subclass of "UIViewController"

Step 3: Create a new class called "ChatScene" in the "Views/Scenes" folder. This is a subclass of "UIController" -- which you created above. Don't forget, you need to override "viewDidLoad" and "didReceiveMemoryWarning" when creating a view controller subclass.

Step 4: Modify 'AppDelegate.swift' so that the 'ChatScene' is the view controller that loads first when we start the app.

Step 5: In your 'UIController' add a line called "view.backgroundColor = UIColor.blue" This is a great demonstration of how Object Orientated Programming is good. Up until now, when we would create a new ViewController, we would always have to change the background color, because the default color is black. Now, whenever we create a new ViewController class, it will be a subclass of "UIController" -- and any default behavior we want will be implemented in this class. We will not need to set "view.backGroundColor" again. We will continue to build out this class as the project continues.

Step 6: Add a 'MessageTableView' to the 'ChatScene' view. The procedure for doing this is identical to how we've added all views to our ViewControllers so far: create the object, give it a frame, set a background color so we can see it, and add it to the view of the viewController (view.addSubview)

Step 7: Implementing a TableView control is going to introduce some new concepts, syntax and types!

We're going to stop the class when everyone get's to Step 7 -- as I need to explain Swift Protocols and Swift Delegates. These are a bit tricky to wrap your head around if you haven't been exposed to these concepts before.

**New Syntax: How to add a Protocol to a Swift Class:**

class ChatScene : UIController, UITableViewDelegate, UITableViewDataSource

{

}

UIController is our Super Class. UITableViewDelegate and UITableViewDataSource are protocols. Protocols are undefined class definitions, and their purpose is to require an implementation of specific functions inside a class. A 'delegate' is a type of communication, it's a mechanism we use to communicate with and control our classes from other parts of our programs.

There are many different ways to implement a TableView. It is one of the most common controls used in IOS apps, and as a result there are many different flavors to accomplish this.

**As soon as you add "UITableViewDelegate" XCode is going to complain that you have not implemented the Protocol functions. We need to add the following functions to our ChatScene class to make it compliant wit the "UITableViewDataSource" Protocol**

func numberOfRows(inSection section: Int) -> Int {

return 1

}

func tableView(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return 1

}

func numberOfSections(in tableView: UITableView) -> Int {

return 0

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell

{

return UITableViewCell()

}

**All this code does is stop XCode from complaining. We have to modify it to get it to do anything interesting. This is where the 'delegate' comes into play. Don't worry about having a deep understanding of what the delegate is, or how it works, as we will be using them extensively.**

**For right now, add this code to the 'viewDidLoad' function:**

messagesTable.delegate = self

messagesTable.dataSource = self

**This set's the ChatScene as both the 'delegate' and the 'dataSource'. Later on, we'll change these variables to reference other classes that we'll create.**

**There is another piece of code that is important -- and this is unique and specific to TableViews:**

messagesTable.register(UITableViewCell.self, forCellReuseIdentifier: "cell")

**a 'cell' is a line in the TableView. These are highly customizable. You can create different types of cells and use them in the same TableView, if you wanted to.**

**New Syntax: Arrays**

All TableView's use a datasource. The datasource is simple an object that stores the information that is displayed in the TableView. We're going to use an array.

var names: [String] = ["Hermione", "Harry", "Ginny", "Ron"]

**Add this array to the 'ChatScene'**

**Finally, modify the Protocol functions:**

func numberOfSections(in tableView: UITableView) -> Int {

return names.count

}

func tableView(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell

{

let cell: UITableViewCell = messagesTable.dequeueReusableCell(withIdentifier: "cell")!

let bkColor = UIColor(red: 196/255, green: 209/255, blue: 148/255, alpha: 1.0)

cell.textLabel?.text = self.names[indexPath.section]

****cell.backgroundColor = bkColor

cell.textLabel?.font = UIFont(name: "Arial", size: 26)

return cell

}

**Challenge: Take the "TableView" code out of the "ChatScene" class and put it into the "MessageTableView" class, but get it to behave the same.**

**Other Tasks:**

1) Add a "addBackgroundImage" function to UIController that accepts a String as a parameter

2) Add a "createButton" function to UIController

3) Add a "createStaticText" function to UIController

4) Add a "createStaticImage" function to UIController

5) Create new Model class called "CommonScreenPoints" in your Models folder that as the following enum: TOP, CENTERTOP, RIGHTTOP, LEFTCENTER, CENTERSCREEN, RIGHTCENTER, BOTTOM, LEFTBOTTOM, RIGHTBOTTOM. Create a function in this class that accepts this Enum as a parameter, and returns an CGPoint. Create a new function called createRect() that accepts the enum, accepts a width and height as parameters. Use your own naming conventions as desired.

**New Syntax Required for #5) switch statements**

switch(controlState)

{

case .add\_ITEMS:

// Add an item into the database

break;

case .change\_PRICES:

break;

case .expire\_ITEMS:

break;

case .perform\_INVENTORY:

break;

case .none:

break;

case .live\_REFILL:

break;

case .theft\_DETECTION:

break;

case .set\_BINS:

break;

}  
  
Refresher:

enum ControlState {

case change\_PRICES

case theft\_DETECTION

case perform\_INVENTORY

case add\_ITEMS

case expire\_ITEMS

case live\_REFILL

case none

}