Advanced Mobile Application Development
Week 2: Tab Bar Controllers

Music

Create a new project and choose the Tabbed Application template.
Let's call it music and make it an iPhone app.
Look at the files created, the Tabbed Application template did a lot for us.
MainStoryboard already has a tab bar controller set up with three scenes.
Click on the tab bar controller. In the identity inspector you'll see that the class is UITabBarController.
In the attributes inspector you'll see that Is Initial View Controller is checked. And in the Connections inspector you'll see that segues are already set up to the other two view controllers.
Click on the first view and look at the identity inspector. It's already set up to use the class FirstViewController and the swift file was created for us.
In the Connections inspector note that a relationship segue has already been set up from the tab bar controller.
You'll also notice that the tab bar item is set up to use an image already in the project called first.png (click on the tab bar button and go into the attributes inspector). We can change the image later.
Look at the second scene as well. This one uses the SecondViewController class, has a relationship segue, and its tab bar item uses an image called second.png.

Go ahead and run it and you'll see that the tab bar is already working and loads both views, so this template does a lot for us.

Single component Picker

In our first view we're going to add a single component picker where you choose a genre of music.
Go into the Storyboard and add delete the two labels there.
Add a picker view(ignore the default data it shows)
Add a label above it that says Music Choser.
Then another label below the picker that we'll use for output.

Create outlet connections for the picker and the 2nd label called musicPicker and choiceLabel. Make sure you're connecting to FirstViewController.swift.

With the picker selected go into the connections inspector and connect the dataSource and delegate to the view controller by clicking in the circle and dragging the connection to View Controller (square in a yellow circle icon on top of the view). Now this picker knows that FirstViewController is its data source and delegate, and will ask it to supply the data to be displayed. (can also do this programmatically in viewDidLoad)

Go into FirstViewController .swift and add the protocol we're going to implement

```
class FirstViewController: UIViewController, UIPickerViewDelegate,
UIPickerViewDataSource
```

You will have an error until we implement the required methods for `UIPickerViewDataSource`
Look at the delegate classes.

Define an array that will hold the music genres.

```
let genre = ["Country", "Pop", "Rock", "Classical", "Alternative", "Hip
Hop", "Jazz"]
```

Now let's add the methods needed to implement the picker.

```
//Methods to implement the picker
//Required for the UIPickerViewDataSource protocol
func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
        return 1 //number of components
}

//Required for the UIPickerViewDataSource protocol
func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component:
Int) -> Int {
        return genre.count //returns number of rows of data
}
```

the delegate methods are optional, but you need to implement at least one and it's often this one.

```
//Picker Delegate methods
//Returns the title for a given row
func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent
component: Int) -> String! {
        return genre[row]
}

//Called when a row is selected
func pickerView(pickerView: UIPickerView, didSelectRow row: Int, inComponent
component: Int) {
        choiceLabel.text="You like \(genre[row])"
//writes the string with the row's content to the label
}
```

Run it in the simulator
Click and drag your mouse in the simulator to move the picker.
Try Resolve Auto Layout issues | All Views in First Controller
Tweak these so the view looks good in portrait and landscape.
Assistant editor – use the jump bar to access preview mode.

Multi-component picker

Either modify the view we've been working in or use the second tab for a 2 component independent picker.
We're going to add a 2nd component that lists the decade for music.
Genre will be component 0
Decade will be component 1

Let's add another array to hold the decade.

```
let decade = ["1950s", "1960s", "1970s", "1980s", "1990s", "2000s", "2010s"]
```

Most of the changes will be in the data source methods.
This time we're going to return 2 in `numberOfComponentsInPickerView` since we have 2 components.

And now we have to check with component is picked before we can return the row count.

```
func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component:
```

```
Int) -> Int {
        if component==0 {
            return genre.count
        }
        else {
            return decade.count
        }
}
```

Then in our delegate method we also have to check which component was picked before we can return the value.

```
func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent
component: Int) -> String! {
        if component==0 {
            return genre[row]
        }
        else {
            return decade[row]
        }
}
```

Now we have to use both components when printing out our results.

```
func pickerView(pickerView: UIPickerView, didSelectRow row: Int, inComponent
component: Int) {
    let genrerow = pickerView.selectedRowInComponent(0)
    let decaderow = pickerView.selectedRowInComponent(1)
    choiceLabel.text="You like \(genre[genrerow]) from the
\(decade[decaderow])"
}
```

You don't need to change anything in the storyboard because all the connections are there.
You might need to make the label field larger since our output is larger.

Snapshot: Multi component picker

Dependent multi-component picker

Now let's add a third tab to our tab bar. In this tab we're going to create a new 2 component picker but this time the components are going to be dependent on each other.
Go back into MainStoryboard and drag a View Controller from the object library onto the canvas.
Make the connection from the tab bar controller to the new view controller by cntrl-click and drag from the tab bar controller to the new view controller and choose Relationship segue view controllers. This will add a third tab bar button that is hooked up to the new view controller.

Now we need a new class to control this view.
File | New | File
iOS Source | Cocoa Touch class
Call it ThirdViewController and make sure it's a subclass of UIViewController.
Uncheck Also create xib file and make sure the language is Swift.
Make sure it's being saved into your project folder and the music target is checked
This should create ThirdViewController.swift

In the Storyboard click on the third ViewController and in the identity inspector change its class to ThirdViewController.

Go into the Storyboard and create a similar interface as before - a picker and two labels (one that says Artist Picker and another label that's empty that we'll use for output). Same constraints will work.
Create outlet connections for the 2nd label and the picker called choiceLabel and artistPicker.
With the picker selected go into the connections inspector and connect the dataSource and delegate to View Controller.

Few applications use arrays you define in the code so this time we're going to use the data stored in a plist.
Copy the artistalbums plist into your project and make sure you choose Copy items if needed and that the project target is checked.
Look at the plist. Note that it's a dictionary, keys are strings, values are arrays of strings.

Go into ThirdViewController.swift and add the protocols like last time.

```
class ThirdViewController: UIViewController, UIPickerViewDataSource,
UIPickerViewDelegate
```

Let's define 2 constants for the component numbers since we'll be using them a lot.

```
    let artistComponent = 0
    let albumComponent = 1
```

Now we're going to define a Dictionary to load our plist into and 2 arrays to hold the artists names and album names.

```
    var artistAlbums : [String: [String]]!
    var artists : [String]!
    var albums : [String]!

override func viewDidLoad() {
            super.viewDidLoad()
        // Do any additional setup after loading the view.
        //use a NSBundle object of the directory for our application to
retrieve the pathname of artistalbums.plist
        let path = NSBundle.mainBundle().pathForResource("artistalbums",
ofType: "plist")
        // use NSDictionary to load the plist and cast to a Dictionary
        artistAlbums =  NSDictionary(contentsOfFile: path!) as!
[String:[String]]
        // asign all the Dictionary keys in the artists array
        artists = Array(artistAlbums.keys)
        // assign all the albums for the first artist in the albums array
        albums = artistAlbums[artists[0]]! as [String]
}
```

The next 3 methods are almost identical to our last project, we're just using the constants we defined.

```
//Required for the UIPickerViewDataSource protocol
    func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
        return 2 //number of components
    }
```

```
    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent
component: Int) -> Int {
        if component == artistComponent {
        return artists.count
    } else {
        return albums.count
        }
    }

//Picker Delegate methods
//Returns the title for a given row
    func pickerView(pickerView: UIPickerView, titleForRow row: Int,
forComponent component: Int) -> String! {
        //checks which component was picked and returns the value for the
requested component
        if component==artistComponent {
            return artists[row]
        }
        else {
            return albums[row]
        }
    }

//Called when a row is selected
    func pickerView(pickerView: UIPickerView, didSelectRow row: Int,
inComponent component: Int) {
        //checks which component was picked
        if component == artistComponent {
            let selectedArtist = artists[row] //gets the selected artist
            albums = artistAlbums[selectedArtist] //gets the albums for the
selected artist
            artistPicker.reloadComponent(albumComponent) //reload the album
component
            artistPicker.selectRow(0, inComponent: albumComponent, animated:
true) //set the album component back to 0
        }
        let artistrow = pickerView.selectedRowInComponent(artistComponent)
//gets the selected row for the artist
        let albumrow = pickerView.selectedRowInComponent(albumComponent)
//gets the selected row for the album
        choiceLabel.text = "You like \(albums[albumrow]) by
\(artists[artistrow])"
}
```

Icons

Let's set the tab bar name and image. Drag the png files into Assets.xcassets that you want to use. Then in the Storyboard click on the tab bar button for the first scene and in the attributes inspector you can change its title and image, or remove the title if you just want an image. (65-note, 120-headphones, 194-note-2, 31-ipod)

(you can download glyphish icons to use)

For app icons you need 120x120 for the 2x and 180x180 for 3x.

You should also set up a launch screen and don't forget constraints.

iTunes

Now let's add a fourth tab to our tab bar. In this tab we're going to access iTunes
Add a new view controller into your storyboard and connect the tab bar controller with a relationship segue.
Then add a new Cocoa Touch class, subclass UIViewController and call it FourthViewController.
Make this the class for your new view controller.
(these steps are the same as what we did when we added the third view controller)

Now we need a new class to control this view.
File | New | File
iOS Source | Cocoa Touch class
Call it FourthViewController and make sure it's a subclass of UIViewController.
This should create FourthViewController.swift

In the Storyboard click on the fourth ViewController and in the identity inspector change its class to FourthViewController.

Now lets get our fourth view set up.
Let's make this a view where a button will take the user to the iTunes music library or if that's not on their device(simulator) then it opens itunes in Safari.
And add a button that says Listen and connect it as an action to a method called gotomusic.
Since this is the fourth scene we must connect to FourthViewController.swift.
Add Missing Constraints for all views in the fourth view controller and fix as needed.
Change the tab bar button image(ipod)

Now let's go into FourthViewController.swift and implement gotomusic().

```
    @IBAction func gotomusic(sender: UIButton) {
    //check to see if there's an app installed to handle this URL scheme
        if(UIApplication.sharedApplication().canOpenURL(NSURL(string:
"pandora://")!)){
           //open the app with this URL scheme
            UIApplication.sharedApplication().openURL(NSURL(string:
"pandora://")!)
        }else {
            if(UIApplication.sharedApplication().canOpenURL(NSURL(string:
"music://")!)){
                UIApplication.sharedApplication().openURL(NSURL(string:
"music://")!)
            } else {
                UIApplication.sharedApplication().openURL(NSURL(string:
"http://www.apple.com/music/")!)
            }
        }
    }
```

For a Swift app, iOS creates a UIApplication object to set up the app's runtime environment: its use of the display, its ability to handle touches and rotation events, etc. This object is also how we can interact with the rest of the iOS system. We're using the sharedApplication() method to get access to another app.
UIApplication also has a UIApplicationDelegate object, which is informed of major life-cycle events that affect the app. This is the AppDelegate class that Xcode gave us to start with. When all the app setup is done, the application(didFinishLaunchingWithOptions:) method gets called. From our point of view, this is where the app "starts," although a bunch of stuff has already been done for us by this point.

UIApplication has the method openURL(), which will launch other applications and have them deal with the NSURL.
mailto:, facetime:, and tel: open the Mail, FaceTime, and Phone apps, respectively.

Starting in iOS9 you have to declare any URL schemes of non-Apple apps you want your app to be able to call canOpenURL(). This does not affect openURL().
Go into Info.plist and add LSApplicationQueriesSchemes as an Array and give it an item "pandora".