ATLS 4120: Mobile Application Development
Week 3: UIKit Framework

Frameworks

- Cocoa Touch includes frameworks for incorporating technologies, services, and features into your apps
- Frameworks provide the code and resources needed in your development

Foundation Framework

- The Foundation framework is the foundational toolkit for iOS programming
- Provides dozens of classes and protocols for a variety of purposes

UIKit Framework

- The UIKit Framework provides the classes needed to construct and manage an application's user interface for iOS https://developer.apple.com/documentation/uikit
  - Creating common UI elements
  - Drawing to the screen
  - Event handling
  - Present text and web content
  - Handle touch and motion based events
- Apple developed a set of user interface controls that work well on the iPhone and iPad.
- Can you list 3 user interface objects that we're used to using on the Web that you don't see in iOS apps?
  - checkbox, radio button, and drop down/select list are not in UIKit
- UIView class https://developer.apple.com/documentation/uikit/uiview (User Interface Views and Controls)
  - A view is the rectangular area on the screen
  - Views display content
  - Views handle touch events (notice it inherits from UIResponder)
  - Hierarchical
    - One superview
    - May have many (or none) subviews
    - Hierarchy is usually constructed in Interface Builder
    - Subviews are displayed on top of its parent view
  - UIWindow class
    - Subclass of UIView
    - Apps only have a single window
    - It's created automatically at run time
  - UIImageView (class reference)
  - UILabel (class reference)
- Controls
  - UIControl (class reference)
    - All iOS controls are subclasses of the UIControl class
    - Can't use UIControl directly
  - UIButton (class reference)
  - UISwitch (class reference)
    - A switch is used to turn on and off a setting or feature.
    - It toggles between Boolean values **true** and **false**
    - **isOn** property is of type Bool that is **true** when it's on otherwise it's **false**.

    - UISlider (class reference)
        - A slider is used to specify a numeric value within a range of values.
        - **value** property is a Float that stores the value.
    - UISegmentedControl (class reference)
        - A segmented control is a multiple choice control that allows users to select a single value.
        - Behaves similarly to a radio button.
        - **selectedSegmentIndex** is a property that stores an Int for the selected index. Starts at 0.
        - Plain, bordered, bar, bezeled
    - UITextField (class reference)
        - Displays an editable one line text area
    - And others that we'll be exploring this semester
- Interface controls are used in three modes
    - Passive
        - store a value but don't trigger an action
        - Text field
    - Active
        - trigger an action
        - buttons
    - Static
        - Inactive, user can't interact with it
        - Image view, labels
- Most UI controls can act in all three modes

There are many other frameworks as well.

- Audio, video, mapping, graphical, etc.

Human Interface Guidelines

Apple's Human Interface Guidelines provide information on designing and building iOS apps. Following these guidelines provide a consistent look and feel for iOS apps and must be adhered to for apps to get approved for the app store.

Interface Essentials https://developer.apple.com/design/human-interface-guidelines/ios/overview/interface-essentials/

There are guidelines for how to use the controls in UIKit.

Labels https://developer.apple.com/design/human-interface-guidelines/ios/controls/labels/

Switch https://developer.apple.com/design/human-interface-guidelines/ios/controls/switches/

Slider https://developer.apple.com/design/human-interface-guidelines/ios/controls/sliders/

Segmented control https://developer.apple.com/design/human-interface-guidelines/ios/controls/segmented-controls/

We'll be talking more about app design next week.

Beatles

(beatles)

New Project, Single View application.

We will use 3 images of close to the same size, around 400x400.

Select the Assets.xcassets item and drag your images in (or click the plus button in the lower-left corner of the editing area and select New Image Set to create a spot for your image).

Copy the main picture into the 3x spot(Beatles_Abbey_Road.png) and rename the image to beatles_abbey_road. (you should also create one for 2 x and 1x).
Repeat the process for beatles1.png and beatles2.png and rename those to beatles1 and beatles2 respectively. We're giving them each a unique name, so we can refer to it elsewhere in the project.

Go into the Main storyboard.
Add an image view. It will try to scale to fit the whole view. Don't worry about the size yet.
In the attributes inspector choose Beatles_Abbey_Road for the image.
Click off the image view and then select it again.
Size it by Editor | Size to Fit Content. This changes the size of the image view to be the size of the image.
Once it's resized, move it to be centered at the top of the view.
In the attributes inspector under View make sure Content Mode is set to Aspect Fit. This will help with your other images if they don't all have the same aspect ratio it will make sure it keeps their aspect ratio.

Add a label right under it and change the text to say The Beatles.

Add a segmented control with two segments named Early 60s and Late 60s.
Look at the attributes inspector and note the segments are numbered 0 and 1.

Connections
What connections will we need? Outlets? Actions?
Connect the image and label as outlets since we'll be changing the image and the text in the label.
Open the assistant editor and make the connections to ViewController.swift.
Name the image beatlesImage, the rest is fine.
Name the label titleLabel, the rest is fine.
This should add in the ViewController.swift file

```
    @IBOutlet weak var beatlesImage: UIImageView!
    @IBOutlet weak var titleLabel: UILabel!
```

The segmented control needs to be an outlet so we can access which segment is selected.
Connect the control as an outlet called imageControl.
It also needs to be connected as an action so we can change the image when the user taps a segment.
Then make an Action connection named changeInfo, type UISegmentedControl, event is value changed.
This created in the swift file

```
    @IBOutlet weak var imageControl: UISegmentedControl!
    @IBAction func changeInfo(_ sender: UISegmentedControl) {
    }
```

Now let's implement the method so different images show depending on when segment is chosen.

```
    @IBAction func changeInfo(_ sender: UISegmentedControl) {
        if imageControl.selectedSegmentIndex==0 {
            titleLabel.text="Young Beatles"
            beatlesImage.image=UIImage(named: "beatles1")
        }
        else if imageControl.selectedSegmentIndex==1 {
            titleLabel.text="Not so young Beatles"
            beatlesImage.image=UIImage(named: "beatles2")
        }
    }
```

I really don't want either segment to be chosen initially. Go into the storyboard, select the segmented control and in the attributes inspector chose Segment 0 and uncheck Selected. If neither segment is selected then neither will be initially selected.

Run your app and make sure the segmented control is changing the image and label.

Switch

Go back into the storyboard and add a switch. Set State to Off.
Put a label next to it that says Capitalization so the user knows what the switch does.
How should we connect the switch?
Connect the switch as an Outlet called capitalSwitch so we can easily access its value.
Now connect it as an Action and name it updateFont, type UISwitch, event Value Changed.
Go into the swift file to implement the method.

```
    @IBAction func updateFont(_ sender: UISwitch) {
        if capitalSwitch.isOn {
            titleLabel.text=titleLabel.text?.uppercased()
        } else {
            titleLabel.text=titleLabel.text?.lowercased()
        }
    }
```

`(capitalized()` does mixed case)

Slider

Add a slider to control the font size so a min of around 6 and a max of 22 is good. Initial can be 16 or something mid-sized. Change these in the attributes inspector.
Add a label next to it to show the font size value. You can use that initial value in the label field.
How should we connect the switch?
Connect the label as an outlet called fontSizeLabel.
Connect the slider as an action called changeFontSize, type UISlider, event Value Changed.

Now implement the method.

```
    @IBAction func changeFontSize(_ sender: UISlider) {
        let fontSize=sender.value
        fontSizeNumberLabel.text=String(format: "%.0f", fontSize)
        let fontSizeCGFloat=CGFloat(fontSize)
        titleLabel.font=UIFont.systemFont(ofSize: fontSizeCGFloat)    }
```

Create a constant called fontSize with the UISlider value.
The text label text property expects a String so we use the String class format initializer to convert the slider value which is a float to a string.
To change the font of the label we have to create a UIFont object with the new font size.
The UIFont class method systemFont(ofSize:) returns a UIFont object.
We cast the slider value which is a float to a CGFloat which is the required parameter type for systemFont(ofSize:). This returns a UIFont object that we assign to the font property of titleLabel.

Now you'll notice that almost everything works but if you change the image then we lose the caps setting. That's because we're changing the label's text and not taking case into account.

We don't want to rewrite the code to check the caps switch so instead we're going to separate out that code into helper methods that we can easily call whenever we need them.

So let's move the image code into a helper method `updateImage()` and the caps code into a method `updateCaps()`

```
    func updateImage() {
        if imageControl.selectedSegmentIndex==0 {
            titleLabel.text="Young Beatles"
            beatlesImage.image=UIImage(named: "beatles1.png")
        }
        else if imageControl.selectedSegmentIndex==1 {
            titleLabel.text="Not so young Beatles"
            beatlesImage.image=UIImage(named: "beatles2.png")
        }
    }

    func updateCaps() {
        if capitalSwitch.on {
            titleLabel.text=titleLabel.text?.uppercased()
        } else {
            titleLabel.text=titleLabel.text?.lowercasd()
        }
    }
```

Then the methods will call those.

```
    @IBAction func changeInfo(_ sender: UISegmentedControl) {
        updateImage()
        updateCaps()
    }
    @IBAction func updateFont(_ sender: UISwitch) {
        updateCaps()
    }
```

Don't forget the launch screen, app icons, and set up constraints for the layout.

Layout

Embed the switch and its label in a stack view. Add spacing as needed (30)
Embed the slider and its label in a stack view. Add spacing as needed (20). (Alignment center)
Then select both those stack views along with the segmented control and embed in a stack view. Add spacing as needed (20). Make sure this stack view has alignment as Fill (using Center messed up the slider).
Add a height constraint for the label (22). This will dedicate space to it so as the font size grows and shrinks it won't change the amount of room it takes and have the effect of moving everything else around.
Select the label and image view and embed those in a stack view. Add spacing as needed (20).
If the image grows, add a width and height constraint.
Now select both stack views and embed those in a stack view. Add spacing as needed (30).
Now we only have the 1 top level stack view to position.
Align center x and align top for the Y position.
Also add a leading and trailing constraint if you want a margin to the edge.

If you add a bottom constraint as well the stack view should grow based on the amount of vertical space. If this stretches out the segmented control or other views, change the distribution of the stack views. (stack view with the segmented control: equal spacing, top stack view: fill proportionally).
Why all these stack views? You'll see that these modular stack views will be helpful when we design a different layout for the compact | compact size class next week.
Look at it on a smaller phone and by changing the spacing values and the top value you should be able to get it to fit.