ATLS 4120/5120: Mobile Application Development
Week 2: OOP

OOP

- Object-Oriented Programming models the natural way humans think about things in terms of attributes and behavior.
- Object-Oriented Programming (OOP) makes building complex, modular and reusable applications much easier.
- OOP combines data and operations on the data (behavior) into one unit, a class.
- Swift and Java are both object oriented languages

Class

- Models a concept or thing in the real world
- Provides a template, or blueprint for its objects.
- Defines the characteristics (data properties) and behavior (methods) of its objects.
  - The data properties provide the class attributes, or characteristics.
  - Similar to functions, methods are the actions or behaviors of the class
- Defines which parts can be seen outside of the class therefore hiding private information
- Analogy: cookie cutter
- Example:

Dog class (slide)

- Breed
- Age
- Owner
- Favorite activity

Objects

- Something that can be acted on
- An instance, or occurrence, of a given class
  - An object of a given class has the structure and behavior defined by the class
  - Many different objects can be defined for a given class
  - All objects of the same class have the same structure
- Properties store data/information
- Methods are actions the object can perform
  - A method should focus on one specific task
  - Just like functions but part of a class
  - Often change the values of properties
  - Instance methods are functions that operate on an object
  - Type methods are functions that operate on the class itself

In real life an object like a chair has properties such as color, material, seating capacity. And it would have functions like reclining, rocking, or maybe vibrating.

Initialization

- Initialization is the process of preparing an object of a class for use.
- Initializers/Constructors are special methods that instantiate a new object of a class.

Encapsulation

- Modular
  - data properties and behavior are packaged into a single well-defined programming unit
- Information Hiding
  - Methods provide the public interface to objects of a class
  - Implementation is kept private

Inheritance

- Inheritance enables classes to form a hierarchy like a family tree.
- Allows subclasses to share the structure and behavior of its superclass.
  - Superclass is the parent class
  - A subclass extends a class
    - Inherits from the superclass
    - Can add properties and methods
    - Can override a method with a new one
- A super class should be generalized and its subclasses become a more specialized definition
- Inheritance allows you to easily reuse code

Example:

Animal class

- Sex
- Type

- Might make sense to move age to the Animal superclass if all animals have an age

The ViewController class created for us to control our view was a subclass of the UIViewController class provided in the SDK.

Why use OOP?

- iOS and Android are based on a OOP architecture
- Encapsulation: data and methods in a cohesive unit
- Defines a public interface through the methods while the rest of the information is private within the object
- eliminates redundancy because code is reusable
- easier to maintain
- easier to debug

CSCI 5828 Foundations of Software Engineering goes more in depth in OOP

Model-View-Controller(MVC)

The MVC architecture is a common design pattern and is used in both iOS and Android (image)

- Model: holds the data and classes
  - Should be UI independent
- View: all items for the user interface (objects in IB)
- Controller: links the model and the view together. The backbone or brain of the app.
  - usually subclasses from UI frameworks that will allow the view and the model to interact
  - Controllers are usually paired with a single view, that's where ViewController comes from
- The goal of MVC is to have any object be in only one of these categories.
  - These categories should never overlap
  - You should be able to change the model and not have to change the UI, and vice versa
- Ensures reusability

daVinci

File | New | Project
iOS Application: Single View Application
Product name: daVinci
Team: None
Organization name: Your name
Organization identifier: ATLAS or something else
Language: Swift

Device: iPad (or you can do iPhone if you prefer)
Leave Use Core Data, Include Unit Tests, and Include UI Tests all unchecked.
Next
Choose a folder for all your iOS projects.
Leave create local git repository unchecked.
Create.

We're going to be using 3 images in this app, all of them with width 440, height 598 (daVinci.png, daVinci_MonaLisa.png, daVinci_Virtruvian.png)
Find/edit/create your images (use Photoshop if needed) and save them as png files.

Select the Assets.xcassets item and click the plus button in the lower-left corner of the editing area. This brings up a small menu of choices, from which you should select New Image Set. This creates a new spot for adding your image file.
Copy the main picture into the 3x spot(daVinci.png) and rename it from Image to DaVinci (you should also create one for 1x and 2x).
Repeat the process 2 more times for the other two images renaming them to MonaLisa and Vitruvian.
Make sure they each have a unique name, so we can refer to it elsewhere in the project.

Click on Main.storyboard
Let's make our initial device view iPad Pro 9.7 which is the same size class as an iPad Air.
Use the – if you need to zoom out.
Go into the Object library (square with a circle)
Scroll or search for label
Drag a Label from the library into the View.
Change the label text to say "Leonardo da Vinci"

Now drag a button into the view.
Change the text to "Paintings" and go into the Attributes inspector under View make Tag=1.
Now add another button and give it a title "Drawings" and tag=2. (or command D to duplicate the first one)

In the object library select an image view and drag in onto the view. It might resize it, don't worry.
Select the image view and go into the size inspector to make the width 440 height 598.
Once it's resized, move it to be centered and under the buttons.
In the attributes inspector under View in Mode choose 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.

At the top of the attributes inspector tab and in the image field choose your main image file (daVinci)

Now let's make the buttons do something. Depending on which button the user presses, a different image and text will show up.
Now we have to connect the interface and the code.
We also need to see ViewController.swift so open up the assistant window. (middle editor button)
Click on the first button and then hold down the control key.
Then click and drag from the button over to the swift file.
Notice the blue fishing line being drawn between these two.
Move your cursor between the curly braces for the class.
When you see a grey box appear release the mouse button.

This window lets you set up the connection between the button and your code.
Connection: Action
Name: chooseArt
Type: UIButton
Event: Touch Up Inside is the standard event to use for buttons.
Arguments: Sender
Now hit Connect.
You should now see in the swift file

```
    @IBAction func chooseArt(_ sender: UIButton) {
    }
```

This is the method header for chooseArt that will be called when the user taps the button.

Now let's connect the second button.
We could create a new method as well, but we can use the one we already have.
Make sure the swift file is open in the assistant editor.
Cntrl click on the second button and drag the blue line to the swift file.
This time go to the method you already have, chooseArt, until it's highlighted in blue and you see a grey popup saying Connect Action. When you see that, release the mouse button to connect the button to chooseArt.

Since we'll want the image to change based on which button is pressed, let's hook up the image as an Outlet and call it artImage.
Control-click from the image to the swift file to make the connection.
Connection: Outlet
Name: artImage
Type:UIImageView
Leave storage as weak.
Connect.
In your swift file it should have added

```
    @IBOutlet weak var artImage: UIImageView!
```

Let's look at these connections a little more.
Click on the View Controller icon and go into the connections inspector and you can see your connections and that both buttons are connected to chooseArt and your image is connected as an outlet.
You'll also see your View is connected, that should ALWAYS be there.
I'll show you how to fix any misspelled connections, for now, just leave them.

Outlets and Actions

Outlets are special properties that point to an object in an Interface Builder file.
Go into ViewController.swift and you'll see @IBOutlet
IBOutlet is a keyword that tells IB that this instance variable is an outlet that will connect to an object in a storyboard
IB will let you make connections *only* to **IBOutlet** instance variables.

Actions are methods in your controller class.
IBAction is a keyword that tells IB that this method is an action that can be triggered by a control.
IB will let you make connections *only* to **IBAction** methods.

Now we're ready to implement the method for the button. Go into ViewController.swift

```
    @IBAction func chooseArt(_ sender: UIButton) {
        if sender.tag == 1 {
            artImage.image=UIImage(named: "MonaLisa")
        }
        else if sender.tag == 2 {
            artImage.image=UIImage(named: "Virtruvian")
        }
    }
```

`sender` is the name of the internal parameter for this method of type UIButton.
Click the option button and hover over UIButton and click to go to the class reference.
The UIButton class has a tag property which stores the tag of the button that you assigned in Interface builder.
The image property is of class UIImage.
It is initially set when we chose our main image in IB.
When the user taps a button we want to change that image.
In the class reference go to Symbols and look at the different init() methods available.
Look at `init?(named: String)`
Strings are always in quotes and must match the filename exactly.
Note that png files don't need a file extension, all other file types do.
We initialize a new UIImage object with the name of the file we want to use and then assign it to the image property of artImage to change the image.

Another way to do this would be to look at the button's title and use that in our conditional statement.
Can anyone find what property holds the button's title? (currentTitle)
If your app might be localized you don't want any code specific to one language, so tags are safer.

<u>Connections</u>
Show common connection errors and how to fix them.
Delete the variable artImage(comment out references to it) and show the run time error.
Delete the variable and the connection and make a new connection by creating a new variable first and then dragging from the circle to the image. A filled in circle means it's connected.
```
@IBOutlet weak var image2: UIImageView!
```
Show the connections inspector and that the connection looks the same. Either way works.

<u>Launch screen</u>
Apps have a launch screen storyboard too.
Click on LaunchScreen.storyboard and for now just add a label with some text and center it both horizontally and vertically. We'll learn how to use images soon.

<u>Finishing Touches</u>
In the simulator hit the home button to go back to the home screen. (Hardware | Home)
We need an icon!
In the Project Navigator click on the Assets.xcassets folder.
You'll notice 4 spots, for notification, settings, spotlight, and app.
For an iPhone project you'll see 60 pt. pt is points, not pixels. Depending on the resolution there is a different ratio between points and pixels. On early iPhones 1 pt=1 px but On most of the later devices

with a Retina display, a single point is actually a 2 × 2 pixel square. On the iPhone 6 Plus 1 point is a 3 × 3–pixel square.
So for iPhone you'll need a120x120 pixel image for the 2x icon, and a 180x180 pixel image for the 3x icon.
For this project you'll notice that the sizes are different because this project is for iPad.
Let's just add the app icon for the home screen, you can do the others later.
For the iPad we will need a 76x76 pixel image for the 1x icon, and a 152x152 pixel image for the 2x icon. For the iPad Pro we will need a 167x167 pixel image for the 2x icon.
They MUST be png files.
Drag them into their corresponding spaces.
Run, then go to the home screen and see your app icon!