ATLS 4120/5120: Mobile Application Development
Week 2: Intro to App design

When designing and developing for mobile you need to consider the platform and the users that you're designing for.

Device considerations

- Limited screen size
- Each app has one window
- Limited system resources
  - Apps need to start and quit quickly
- Might not always have Internet access
- Could be interrupted
- No physical keyboard and mouse for interaction
  - The comfortable minimum size of tappable UI elements is 44 x 44 points
  - Support common gestures

User expectations

- Apps should be easy to figure out and use
  - How long do you give a new app?
- There are over a million apps, you're trying to convince users to use yours!
- Limit friction
  - Download, install, start instantly, no lengthy startup
  - Avoid requiring initial signup/login
  - Show content immediately
- Avoid unnecessary interruptions

Understand platform conventions

- Apps should be delightful to use
- Apps should be intuitive to use
  - No/limited instructions or help should be needed
  - Take advantage of platform conventions
- Apps should do one thing and do it well
  - Not a "multi-purpose" web site
  - How many apps does ESPN or Google have?
- No "Home"
- Scrolling is ok
- Apps are often interrupted

Design for the platform

- Use layout that makes sense for the device
  - iPad apps shouldn't just be bigger
- Integrate aesthetics with function
- Artwork and image should be useful and draw the user in
  - Adapt art to the screen size, high quality media is expected
  - No logos other than your app icon and possibly splash screen
- Handle different orientations
  - Kindle app supports both portrait and landscape on tablets but only portrait on phones
- Focus on a streamlined flow for the app's purpose

    - Universal apps should have a consistent flow
- iOS and Android have different platform conventions. Embracing these makes app feel "right" and what users are expecting

<u>Content</u>

- Provide only relevant, appropriate content that's useful to the immediate task
- Don't overshadow the content
- Avoid including everything you can think of
- If in doubt, leave it out
- Focus on the goal and the user

<u>Focus on the User</u>

- Target apps to a specific user level
- Put the users in control
- Get them to the relevant information quickly
- Provide a streamlined flow for the app's goal
- Make sure that all user input is valuable
- Provide subtle but clear, immediate feedback
- Create a compelling user experience
    - User interaction consistency

Focus on the goal of your app and the problem you're trying to solve. We'll be talking more about this as you start thinking of your first app.

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
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 drag your images in.
You can also 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 eventually 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 use the iPhone Xr view for our storyboard.
Use the – if you need to zoom out.
Go into the Object library (square with a circle) (option click for it to remain open)
Scroll or search for label
Drag a Label from the library into the view's safe area.
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.
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. At the top of the attributes inspector tab in the image field choose your main image file (daVinci)
In the attributes inspector under View in Mode choose Aspect Fit. This will ensure that as you resize the image the aspect ratio won't be changed. It will also help with your other images if they don't all have the same aspect ratio it will make sure it keeps their aspect ratio.
Select the image view and resize it or use Editor | Size to fit content.
Then resize the image view so it fills up the width of the safe area. (mine has width 374, height 506).
Once it's resized, move it to be centered and under the buttons.

Now let's make the buttons do something. Depending on which button the user taps, a different image will be shown.
Now we have to connect the interface and the code.
We 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 drag to the method you already have, chooseArt, until it's highlighted in blue with a grey box 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.

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.
The UIButton class has a tag property which stores the tag of the button that you assigned in Interface builder.
Click the option button and hover over UIImageView (the type of artImage) and click to go to the class reference. You can read the overview then click on Topics and under accessing the displayed images you see a property called image which is the image displayed in the view.
The image property is of class UIImage. Click on it for more information.
It is initially set when we chose our main image in IB.
When the user taps a button we want to change that image.

Go to the UIImage class reference by either clicking on UIImage or using the search documentation field.
Go to Topics and look at loading and caching images.
Look at `init?(named: String)`
This returns the image object associated with the specified filename.
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.

App Icon
App Icon Design 10 mins (show to 5:20) https://developer.apple.com/videos/play/wwdc2017/822/

- Metaphor
- Simplicity
- Connection
- Lineage

https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/app-icon/
In the simulator hit the home button to go back to the home screen. (Hardware | Home)
We need an app icon!
In the Project Navigator click on the Assets.xcassets folder.
Click on AppIcon
You'll notice place holders for notification, settings, spotlight, app an d app store for iPhone and iPad.
For now let's just add the app icon for the iPhone app home screen, you can do the others later.
60 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 Plus phones 1 point is a 3 × 3–pixel square.
We will need a 120x120 pixel image for the 2x icon(60x2=120), and a 180x180 pixel image for the 3x icon(60x3=180)
They MUST be png files and be exactly the right size.
PNG is the best format to use as Xcode optimizes them to make them the most efficient to use in iOS apps
iOS will automatically add a mask to make it looks like the others.
Drag the 120x120 png file into the app 2x space.
Drag the 180x180 png file into the app 3x space.
Now run it and go to the home screen(Hardware | Home) to see your app icon!
You will see warnings that you're missing iPad app icons. You can add these or change your app from Universal to iPhone in your helloworld target Deployment Info.

Launch screen
https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/launch-screen/
A launch screen appears instantly when your app starts and is quickly replaced with the first view of your app.
Use the LaunchScreen.storyboard for your launch screen.
Click on LaunchScreen.storyboard and for now just add a label with some text and center it both horizontally and vertically.