

ATLS 4120/5120: Mobile Application Development

Week 9: Android Development Intro

Now that we've seen the basic structure of an Android app, let's look at how we can create one that does something.

All Android documentation is on their developer web site <https://developer.android.com> | Docs <https://developer.android.com/reference/> | Android Platform

Views

<https://developer.android.com/training/basics/firstapp/building-ui>

A view is the building block for all user interface components

- The View class is the base class for all widgets **android.view.View**
<https://developer.android.com/reference/android/view/View.html>
 - TextView
 - EditText
 - Button
 - ImageView
 - Check box
 - and many others
- Views that can contain other views are subclassed from the Android ViewGroup class **android.view.ViewGroup** which is a subclass of the View class. Single parent view with multiple children. (ex: RadioGroup is the parent with multiple RadioButtons)
 - Menus
 - Lists
 - Radio group
 - Web views
 - Spinner
 - Layouts
 - and many others
- Access views in your code using **findViewById(id)**
- Android includes many common UI events that you can set up a listener for and assign a handler to. We'll look at them more closely next week.

Views are saved as XML -- eXtensible Markup Language

- XML is a markup language designed to structure data
- XML rules
 - First line - XML declaration
 - XML documents must have a root element
 - Attribute values must be in quotes
 - All elements must have a closing tag
 - Tags are case sensitive
 - Elements must be properly nested
 - XML must be well formed

Widgets

The Android SDK comes with many widgets to build your user interface. We're going to look at 4 today and 5 more next week.

ATLS 4120: Mobile Application Development

Week 9: Android Development Overview

Android

Android 1.0, the first commercial version, was released on 9/23/08.

It now has over 75% of the worldwide market share compared with 20% for iOS. (slide)

In the US iOS has 54% and Android 45%. (slide)

Android Development

Developer web site <http://developer.android.com>

Android Studio (or Download link)

<http://developer.android.com/studio/> (scroll down for system requirements)

- Android development can be done on Windows, Mac OSX, or Linux systems
- Android Studio
 - Integrated Development Environment (IDE)
 - Android Software Developer's Kit (SDK)
 - Emulator to run, test, and debug your apps on different virtual devices
 - Tools and profilers for testing
 - Initially released in 2013, previously the SDK was bundled with Eclipse
 - Can also use the SDK with the command line or another SDK
- Java programming language
 - Although Android uses Java it does not run .class or .jar files, it uses its own format for compiled code
- Announced Kotlin support at Google IO in 5/17 and was integrated into Android Studio 3.0
- eXtensible Markup Language (XML)

As of Android Studio 2.2, the IDE comes bundled with a custom OpenJDK build which contains the Java Development Kit (JDK) and a bunch of additional fixes to make the IDE work better (such as improved font rendering).

Use the SDK Manager (Tools | Android | SDK Manager) to download additional tools and components.

Older versions required you download the Java Development Kit (JDK) and Java Runtime Environment (JRE) separately, but this is no longer required.

Android Setup

The first time you launch Android Studio it will take you to a setup wizard.

Configure | SDK Manager to install other needed SDK components.

OR

Tools | Android | SDK Manager to install other needed SDK components.

SDK Platforms (show package details)

- Android 9.0 (API 28)
- Show Package Details
 - Android SDK Platform
 - Google APIs Intel x86 Atom System Image (for the emulator)

SDK Tools

- Android SDK Build tools
- Android Emulator
- Android SDK Platform-Tools

- Android SDK Tools
- Documentation (do this later)
- Intel x86 Emulator Accelerator (HAXM Installer) (do this later)

Support Repositories (do these later)

- ConstraintLayout for Android
- Solver for ConstraintLayout
- Android Support Repository
- Google Repository
- Google USB Driver (windows only)

SDK Update Sites

All should be checked.

Chose top license, click Accept. Install.

Android Releases

- Platform versions and stats on the Android Developer site dashboard
<http://developer.android.com/about/dashboards>
- Google -> phone manufacturers (OEMs)
- OEMS -> carriers
- Android updates take about 3-6 months for a new version is available on a carrier (slide)
- When creating a new project you need to decide what Android versions to support

Android Devices

Android runs on devices that are all different <http://developer.android.com/about/dashboards> (scroll down)

- Screen sizes
- Processor speed
- Screen density
- The number of simultaneous touches the touch screen can register
- The quantity and positioning of front and back cameras
- Bluetooth

Screen densities https://developer.android.com/guide/practices/screens_support.html
<https://developer.android.com/training/multiscreen/screendensities>

The density-independent pixel is equivalent to one physical pixel on a medium density screen (160 dpi screen).

At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use.

The conversion of dp units to screen pixels is: $px = dp * (dpi / 160)$.

So on a 240 dpi screen, 1 dp equals 1.5 physical pixels.

Android Terminology

- An activity is a single, defined thing a user can do
 - Usually associated with one screen
 - Written in Java or Kotlin
 - Android Studio has activity templates
- A layout describes the appearance of the screen
 - Design view
 - Written in XML

- The emulator lets you define Android Virtual Devices(AVD)
 - A device configuration that models a specific device
- Simulators imitate the software environment of the device but not the hardware.
 - They have access to all the host(Mac) hardware's resources
- Emulators imitate the software AND hardware environments of the actual device
 - A flight simulator simulates a flight. If it was a flight emulator then you'd actually be transported to the destination.

Android Studio

<https://developer.android.com/studio/projects/create-project>

Let's create our first Android app and make sure our environment is all set up.

From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)

Application Name: HelloAndroid

Company domain: a qualifier that will be appended to the package name

Project location: the directory for your project

Package name: the fully qualified name for the project

Don't check either Include check boxes

Form factors: Phone and Tablet (leave others unchecked)

Minimum SDK: API 21: Android 5.0 Lollipop (21 is the minimum API for Material Design)

Add an Activity to Mobile: Empty Activity

Activity Name: MainActivity (we can leave this)

Generate Layout File should be checked

Layout name: activity_main

Uncheck Backwards Compatibility (uses Activity as the base class instead of AppCompatActivity)

(Preferences | Editor | Color Scheme to change scheme to presentation)

Android Studio Tour

<https://developer.android.com/studio/intro/>

The left most pane is the Project Tool Window which provides a hierarchical overview of the project file structure.

The default setting is the Android view which is what I usually leave it in but there are other choices for viewing your project.

Also notice the tabs on the left which will change what is shown in in the left pane.

The middle pane is the editor window which shows the file you're editing. You'll see different editors here depending on what type of file you're editing. If you have more than one file open you will see tabs right above the editor.

Above that is the navigation bar which provides another way to navigate your project files.

When you're editing a layout's xml file the preview window shows on the right side. You can toggle this using the tabs on the far right or hide it.

The bottom window shows the status. There are different tabs at the bottom to choose what you want to see and you can also hide it.

Android Virtual Device (AVD)

<https://developer.android.com/studio/run/managing-avds>

Tools | AVD Manager

Create a virtual device

Category: Phone

Chose a phone like the Pixel - size: large, density: xxhdpi or the Nexus 4 has the size: normal, density: xhdpi

System Image: Pie Android 9.0 or Oreo Android 8 (download the needed system image)

AVD Name: Pixel API 28 (or something else descriptive)

Leave the rest as is and Finish

Now you should see your AVD as a virtual device.

Close the AVD Manager.

Project Files

Now let's take a closer look at what was created in our project. By default, Android Studio displays your project files in the Android view. This view does not reflect the actual file hierarchy on disk, it is simplified for ease of use.

In the Android view you see 3 main areas of your app

1. manifests: Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest file describes the fundamental characteristics of the app and defines each of its components. Every part of your app will be declared in this file. It acts as a blueprint for putting all the pieces of an app together.
 - a. The package name is the unique identifier for your app on devices and the Play store
 - b. The application tag describes various characteristics of your app
 - c. Our activity also has a tag with the name MainActivity
2. java: These are your Java files, using the package name
 - a. Unless you're doing testing, you always want to look in the first one (that doesn't say test)
 - b. MainActivity.java defines a class for us called MainActivity that extends Activity. (AppCompatActivity if you checked backwards compatibility). Extend in Java creates a subclass from a superclass. The Activity class is used for all Android activities.
 - c. Ctrl-Q (PC), Ctrl-J (Mac) opens the documentation for wherever your cursor is. Context sensitive Java and Android documentation can be accessed by clicking on the declaration.
3. res: these are resource files that include
 - a. Drawable – images for the project
 - b. Layout – the layout files for the user interface in xml
 - i. activity_main.xml defines the one TextView that was created for us and its properties including ConstraintLayout properties.
 - ii. Two tabs at the bottom let you switch between text and design mode.
 - c. Mipmap – includes files such as launcher images (app icons are called launcher icons)
 - d. Values – resource files that include values such as the values for strings, styles, colors, etc

The logic is in Java and everything else is a resource.

If you switch to the Project view you'll see the actual file structure of the project. There are a lot of other files, but we're not going to worry about most of them.

R class

I do want to point out just one file because you'll see it referred to.

App/generatedJava/com.example.aileen.helloandroid/R.java

The R.java file acts as an index to all of the resources identified in your project. This file is automatically generated for every Android project and the "R" stands for resources. Any time you change, add, or remove a resource, the R class is automatically regenerated. You should never manually edit the R.java source files.

Also, you're going to see the term Gradle often. Gradle is the build tool in Android Studio. It compiles and builds the app package file known as an APK file.

Run the App

Click the Play button to run your app.

Chose Launch Emulator and chose the AVD you just set up.

You might have to wait a bit for the Emulator to start and launch your app.

The first time an emulator starts it can take a few minutes.

You can leave the emulator running so it will be faster.

You might need to unlock your device – just swipe the padlock icon upwards and you should see the sample app.

Most newer Android devices have 3 virtual buttons.

Left button (back arrow) goes back to the last activity

Middle button (home) takes you to the home screen

Right button show a list running apps (like a double tap of the home button on iOS)

The favorites tray is at the bottom with the all apps button in the center.

Layout

<https://developer.android.com/studio/write/layout-editor>

Open the activity_main.xml file and go into the Design mode.

Underneath the file tabs you can choose between Design view and/or blueprint view.

Moving across to the right is a button that let's you change the orientation.

Then there's a drop down that let's you chose what device you preview, the API, app theme, language, and layout variants.

To the right of those you'll see zoom controls.

Below those you'll see a bunch of controls for Autoconnect. For now just make sure Autoconnect is on (it should be by default) by hovering over the magnet looking icon.

Now take a look in the palette to the left of the layout editor to see all the layouts and widgets available for use.

A text view has already been added for us. Click on it in the Component Tree and all the way to the right open up the Properties pane. Here you can see some of the attributes for the TextView. You can see more using the arrows or clicking on View all attributes.

Look at the XML for TextView by going into Text mode.

You can see all the same properties including the constraints that were automatically added.

String Resources

Notice the textview has `android:text="Hello World!"`

It's not good practice to hard code string values in your xml file, you should use string resources.

strings.xml in the values folder stores all the string resources.

Go into strings.xml and add this resource with some initial value so you can see it worked.

`<string name="text_message">Hello Android </string>`

Back in activity_main.xml change the textview to `android:text="@string/text_message"`

The '@' sign means it's defined as a resource.

Now go into Design mode or Preview and you will see our textview has the new string value.

You can click anywhere on `@string/text_message` and click cmd-B (ctrl B on a pc) to automatically open the strings.xml file where the values for the string resources are stored.

Let's also make the text larger by either scrolling in the Properties pane to textSize or adding in the xml

`android:textSize="48sp"`

We will always use the "sp" unit for font sizes as it allows the font size to scale for the user based on their settings. (scale-independent pixel)

sp has scalable ratio: $sp = px * ratio * scale$. Where ratio never changes, but scale is user configurable.

This will allow the font size to scale such as for people who set larger font sizes in their settings.

If this is a value you're going to be using throughout your layout you should create an entry in the dimens.xml file with a name so you can easily reuse it.

Now let's change the background color through the xml.

In the top level <ConstraintLayout tag before the close tag > add `android:background="#ff6232"`

Notice that a small colored square appears in the gutter to the left. Click on it and it brings up a color wheel. Open up the preview and you'll see the result.

Instant Run

To run your app again in the emulator you could click Run again, or you can use Instant Run.

Instant Run is the button with the lightening bolt to the right of the Run button and can quickly apply small changes to your app already running in the emulator without doing an entire new compile.

Therefore it's always faster, so I recommend using it when you can.

Code completion

The editor in Android Studio has code completion just like in Xcode. As you type you will get selections. To accept the top most suggestion, press the Enter or Tab key on the keyboard. To select a different suggestion, use the arrow keys to move up and down the list, once again using the Enter or Tab key to select the highlighted item.

In the XML file because everything starts with "android" it gets old typing that. You can just start typing what comes after the colon and the code suggestions will still come up. So instead of typing

`android:background` just start typing `background` and you'll see `android:background` as a choice.

To see other suggestions, click on a word and then use ctrl-space and the editor will show you a list of alternative suggestions.

Can you guess how you would change the font color for the textview?

In <TextView> add `android:textColor="#FFFFFF0D"`

Note: To open a project in Android Studio there is no one file you can click on to open the project (similar to the xcodeproj file). To open a project go into Android Studio and open an existing project from there. If you're opening a project that you did not create, such as my samples, chose import project instead to avoid configuration errors.

TextView

- Text views are used to display text `<TextView .../>` **android.widget.TextView**
<https://developer.android.com/reference/android/widget/TextView.html>
- The **android:textSize** attribute controls the size of the text
 - Use the sp unit for scale-independent pixels
 - Scales based on the user's font size setting
- The **setText(text)** method changes the string in the text view
- Not editable by the user

EditText

- Edit text is like a text view but editable `<EditText .../>` **android.widget.EditText**
<https://developer.android.com/reference/android/widget/EditText.html>
- The **android:hint** attribute gives a hint to the user as to how to fill it in
- The **android:inputType** attribute defines what type of data you're expecting
 - Number, phone, textPassword, and others
 - Android will show the relevant keyboard
 - Can chain multiple input types with "|"
- **getText().toString()** retrieves the String

Button

<https://developer.android.com/guide/topics/ui/controls/button.html>

- Buttons usually make your app do something when clicked `<Button .../>`
android.widget.Button <https://developer.android.com/reference/android/widget/Button.html>
- The **click** event is fired when the button is clicked. Set up the listener and handler to respond to the event.

ImageView

- ImageViews display an image **android.widget.ImageView**
<https://developer.android.com/reference/android/widget/ImageView.html>
- Images are added to the res/drawable folder in your project
- You can create folders to hold images for different screen size densities
- The **android:src** attribute specifies what image you want to display
 - @drawable/imagename
- The **setImageResource(resId)** method sets the image source

Resources

A resource is a part of your app that is not code – images, icons, audio, etc

- You should use resources for strings instead of hard coding their values
android:text="@string/heading"
 - Easier to make changes
 - Localization
 - @string indicates it's a string in the strings.xml resource file
 - heading is the name of the string
- Use ids for resources you want to access in your code **android:id="@+id/message"**
 - When you add IDs to resources they will have a + sign because you are creating the ID
 - You don't use the + when you are referencing the resources.
 - **findViewById(id)** uses the id to access the resource

Layouts

Layouts define how you want your view laid out. Constraint layouts were introduced in AS 2.2 in the fall of 2016 while still in beta and they became the default layout when you create a new project starting in 2.3.3 so we will be using them. For now make sure AutoConnect is on so it will handle our constraints for us.

Halloween

From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)

Application Name: Halloween

Company name: a qualifier that will be appended to the package name

Project location: the directory for your project /Users/aileen/Documents/AndroidProjects/Feelings

Package name: the fully qualified name for the project

Form factors: Phone and Tablet (leave others unchecked)

Minimum SDK: API 21: Android 5.0 Lollipop

Add an Activity to Mobile: Empty Activity

Activity Name: MainActivity (we can leave this)

Make sure Generate Layout File is checked

Layout name: activity_main

Backwards Compatibility should not be checked

Open the activity_main.xml file in the Design editor.

The textView that is there should be in the center.

Remove **android:text** property for the textView either in the properties area or in the XML.

The Textview doesn't have an ID so add one so we can refer to this textview in our code.

android:id="@+id/message"

When you add IDs to resources they will have a + sign because you are creating the ID.

You don't use the + when you are referencing the resources.

We will use the ID when referring to it in our code.

Button

In Design mode add a button above the textView. You'll be able to see the textView when you drag over the view. Or switch to blueprint mode.

Look in the xml and see what was added.

Notice it has **android:id**

You'll also notice it has 1 error and 1 warning. Look at what they say. We'll deal with the text first.

The button got created with default text, let's change that.

Add string resources in strings.xml (res/values)

```
<string name="boo">Boo</string>
```

Back in activity_main.xml update the text

android:text="@string/boo"

The second error says that it's not constrained vertically. You can see it has left and right constraints for it's horizontal positioning.

In design view select the button and chose Infer Constraints which is the icon that looks like a magic wand.

This adds a constraint for the bottom margin to the message textView with a value in dp (mine is 50).

Now we want the button to do something. In the button tag start typing onclick and use autocomplete.

android:onClick="sayBoo"

sayBoo is the method we want the button to call in our code when the click event fires.

Now we have to create this method in our MainActivity.java file

Either click on the lightbulb and chose Create sayBoo(View) in MainActivity or just go into MainActivity.java and create it.

Your method must follow the structure **public void methodname(View view) { }**

Android looks for a public method with a void return value, with a method name that matches the method specified in the layout XML.

The parameter refers to the GUI component that triggers the method (in this case, the button). Buttons and textviews are both of type View.

You need to import View and TextView so either add it manually or have AS add it automatically. Android Studio | Preferences | Editor | General | Auto Import | Java and check “Add unambiguous imports on the fly” and “Optimize imports on the fly”

```
import android.view.View;
```

```
import android.widget.TextView;
```

```
public void sayBoo(View view) {  
    TextView booText = findViewById(R.id.message);  
    booText.setText("Happy Halloween!");  
}
```

We create a TextView object called booText.

The findViewById(id) method is how Java can get access to the UI components using their ids.

The R.java class is automatically generated for us and keeps track of all our resources including ids.

R.id.message grabs a reference to our textview. (note that R must be capitalized)

So now our booText object is a reference to our text view.

We can set the text by using the setText() method. You can't use dot notation, you must use getter and setter methods.

Remember your semi colons!

Run it and try it out. Can you make the text larger and the background a fun color?

EditText

Add an EditText (Plain Text) above the button.

It should have an id

```
android:id="@+id/editText"
```

Remove **android:text** as we don't want there to be text in the EditText

In strings.xml add **<string name="name">Name</string>**

In activity_main.xml update the editView.

```
android:hint="@string/name"
```

It might also have android:ems, that determines the width of the text field

There is still one error that it's not constrained vertically. Run it to see what happens if it's not constrained vertically. Then use Infer Constraints and it will add a vertical constraint.

Now let's update MainActivity.java so our message is personalized with our name. Update boo() after TextView.

```
EditText name = findViewById(R.id.editText);
```

```
String nameValue = name.getText().toString();
```

```
booText.setText("Happy Halloween " + nameValue + " !");
```

We create a name object so we have a reference to our EditText.
Then we create a string and use `getText()` to get the text in the EditText. `getText()` returns a value of type `Editable` so we use `toString()` to convert it to a `String` so we can use it in our `TextView`.
Use the `+` to concatenate strings and/or variable values.
Use Instant run to run your app.

In an app where you need access to the UI components throughout the class you can make them global by adding `TextView booText;` right under the class definition.

ImageView

Copy and paste your image into the drawables folder. (ghost.png)
In `activity_main.xml` add an `imageView` below the `textView`.
If it makes you pick an image go ahead and chose ghost.png in the project section.
Move it around so it fits. Notice how the constraint values automatically change.
Remove `app:srcCompat="@drawable/ghost"` so the app starts without an image, we'll assign it programmatically.

Add `boo()` in `MainActivity.java` so our image shows up when the button is tapped.

```
ImageView ghost = findViewById(R.id.imageView);  
ghost.setImageResource(R.drawable.ghost);
```

The `R` class uses the name of the drawable resource so make sure this matches the name you gave it in `AS`.

Move the widgets around in the design view, the constraints will update automatically. With a widget selected you can also easily change the value for a constraint in the attributes panel.
Use Instant Run to quickly see the changes in the emulator.

ATLS 4120: Mobile Application Development

Week 9: Android Development Overview

Android

Android 1.0, the first commercial version, was released on 9/23/08.

It now has over 75% of the worldwide market share compared with 20% for iOS. (slide)

In the US iOS has 54% and Android 45%. (slide)

Android Development

Developer web site <http://developer.android.com>

Android Studio (or Download link)

<http://developer.android.com/studio/> (scroll down for system requirements)

- Android development can be done on Windows, Mac OSX, or Linux systems
- Android Studio
 - Integrated Development Environment (IDE)
 - Android Software Developer's Kit (SDK)
 - Emulator to run, test, and debug your apps on different virtual devices
 - Tools and profilers for testing
 - Initially released in 2013, previously the SDK was bundled with Eclipse
 - Can also use the SDK with the command line or another SDK
- Java programming language
 - Although Android uses Java it does not run .class or .jar files, it uses its own format for compiled code
- Announced Kotlin support at Google IO in 5/17 and was integrated into Android Studio 3.0
- eXtensible Markup Language (XML)

As of Android Studio 2.2, the IDE comes bundled with a custom OpenJDK build which contains the Java Development Kit (JDK) and a bunch of additional fixes to make the IDE work better (such as improved font rendering).

Use the SDK Manager (Tools | Android | SDK Manager) to download additional tools and components.

Older versions required you download the Java Development Kit (JDK) and Java Runtime Environment (JRE) separately, but this is no longer required.

Android Setup

The first time you launch Android Studio it will take you to a setup wizard.

Configure | SDK Manager to install other needed SDK components.

OR

Tools | Android | SDK Manager to install other needed SDK components.

SDK Platforms (show package details)

- Android 9.0 (API 28)
- Show Package Details
 - Android SDK Platform
 - Google APIs Intel x86 Atom System Image (for the emulator)

SDK Tools

- Android SDK Build tools
- Android Emulator
- Android SDK Platform-Tools

- Android SDK Tools
- Documentation (do this later)
- Intel x86 Emulator Accelerator (HAXM Installer) (do this later)

Support Repositories (do these later)

- ConstraintLayout for Android
- Solver for ConstraintLayout
- Android Support Repository
- Google Repository
- Google USB Driver (windows only)

SDK Update Sites

All should be checked.

Chose top license, click Accept. Install.

Android Releases

- Platform versions and stats on the Android Developer site dashboard
<http://developer.android.com/about/dashboards>
- Google -> phone manufacturers (OEMs)
- OEMS -> carriers
- Android updates take about 3-6 months for a new version is available on a carrier (slide)
- When creating a new project you need to decide what Android versions to support

Android Devices

Android runs on devices that are all different <http://developer.android.com/about/dashboards> (scroll down)

- Screen sizes
- Processor speed
- Screen density
- The number of simultaneous touches the touch screen can register
- The quantity and positioning of front and back cameras
- Bluetooth

Screen densities https://developer.android.com/guide/practices/screens_support.html

<https://developer.android.com/training/multiscreen/screendensities>

The density-independent pixel is equivalent to one physical pixel on a medium density screen (160 dpi screen).

At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use.

The conversion of dp units to screen pixels is: $px = dp * (dpi / 160)$.

So on a 240 dpi screen, 1 dp equals 1.5 physical pixels.

Android Terminology

- An activity is a single, defined thing a user can do
 - Usually associated with one screen
 - Written in Java or Kotlin
 - Android Studio has activity templates
- A layout describes the appearance of the screen
 - Design view
 - Written in XML

- The emulator lets you define Android Virtual Devices(AVD)
 - A device configuration that models a specific device
- Simulators imitate the software environment of the device but not the hardware.
 - They have access to all the host(Mac) hardware's resources
- Emulators imitate the software AND hardware environments of the actual device
 - A flight simulator simulates a flight. If it was a flight emulator then you'd actually be transported to the destination.

Android Studio

<https://developer.android.com/studio/projects/create-project>

Let's create our first Android app and make sure our environment is all set up.

From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)

Application Name: HelloAndroid

Company domain: a qualifier that will be appended to the package name

Project location: the directory for your project

Package name: the fully qualified name for the project

Don't check either Include check boxes

Form factors: Phone and Tablet (leave others unchecked)

Minimum SDK: API 21: Android 5.0 Lollipop (21 is the minimum API for Material Design)

Add an Activity to Mobile: Empty Activity

Activity Name: MainActivity (we can leave this)

Generate Layout File should be checked

Layout name: activity_main

Uncheck Backwards Compatibility (uses Activity as the base class instead of AppCompatActivity)

(Preferences | Editor | Color Scheme to change scheme to presentation)

Android Studio Tour

<https://developer.android.com/studio/intro/>

The left most pane is the Project Tool Window which provides a hierarchical overview of the project file structure.

The default setting is the Android view which is what I usually leave it in but there are other choices for viewing your project.

Also notice the tabs on the left which will change what is shown in in the left pane.

The middle pane is the editor window which shows the file you're editing. You'll see different editors here depending on what type of file you're editing. If you have more than one file open you will see tabs right above the editor.

Above that is the navigation bar which provides another way to navigate your project files.

When you're editing a layout's xml file the preview window shows on the right side. You can toggle this using the tabs on the far right or hide it.

The bottom window shows the status. There are different tabs at the bottom to choose what you want to see and you can also hide it.

Android Virtual Device (AVD)

<https://developer.android.com/studio/run/managing-avds>

Tools | AVD Manager

Create a virtual device

Category: Phone

Chose a phone like the Pixel - size: large, density: xxhdpi or the Nexus 4 has the size: normal, density: xhdpi

System Image: Pie Android 9.0 or Oreo Android 8 (download the needed system image)

AVD Name: Pixel API 28 (or something else descriptive)

Leave the rest as is and Finish

Now you should see your AVD as a virtual device.

Close the AVD Manager.

Project Files

Now let's take a closer look at what was created in our project. By default, Android Studio displays your project files in the Android view. This view does not reflect the actual file hierarchy on disk, it is simplified for ease of use.

In the Android view you see 3 main areas of your app

1. manifests: Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest file describes the fundamental characteristics of the app and defines each of its components. Every part of your app will be declared in this file. It acts as a blueprint for putting all the pieces of an app together.
 - a. The package name is the unique identifier for your app on devices and the Play store
 - b. The application tag describes various characteristics of your app
 - c. Our activity also has a tag with the name MainActivity
2. java: These are your Java files, using the package name
 - a. Unless you're doing testing, you always want to look in the first one (that doesn't say test)
 - b. MainActivity.java defines a class for us called MainActivity that extends Activity. (AppCompatActivity if you checked backwards compatibility). Extend in Java creates a subclass from a superclass. The Activity class is used for all Android activities.
 - c. Ctrl-Q (PC), Ctrl-J (Mac) opens the documentation for wherever your cursor is. Context sensitive Java and Android documentation can be accessed by clicking on the declaration.
3. res: these are resource files that include
 - a. Drawable – images for the project
 - b. Layout – the layout files for the user interface in xml
 - i. activity_main.xml defines the one TextView that was created for us and its properties including ConstraintLayout properties.
 - ii. Two tabs at the bottom let you switch between text and design mode.
 - c. Mipmap – includes files such as launcher images (app icons are called launcher icons)
 - d. Values – resource files that include values such as the values for strings, styles, colors, etc

The logic is in Java and everything else is a resource.

If you switch to the Project view you'll see the actual file structure of the project. There are a lot of other files, but we're not going to worry about most of them.

R class

I do want to point out just one file because you'll see it referred to.

App/generatedJava/com.example.aileen.helloandroid/R.java

The R.java file acts as an index to all of the resources identified in your project. This file is automatically generated for every Android project and the "R" stands for resources. Any time you change, add, or remove a resource, the R class is automatically regenerated. You should never manually edit the R.java source files.

Also, you're going to see the term Gradle often. Gradle is the build tool in Android Studio. It compiles and builds the app package file known as an APK file.

Run the App

Click the Play button to run your app.

Chose Launch Emulator and chose the AVD you just set up.

You might have to wait a bit for the Emulator to start and launch your app.

The first time an emulator starts it can take a few minutes.

You can leave the emulator running so it will be faster.

You might need to unlock your device – just swipe the padlock icon upwards and you should see the sample app.

Most newer Android devices have 3 virtual buttons.

Left button (back arrow) goes back to the last activity

Middle button (home) takes you to the home screen

Right button show a list running apps (like a double tap of the home button on iOS)

The favorites tray is at the bottom with the all apps button in the center.

Layout

<https://developer.android.com/studio/write/layout-editor>

Open the activity_main.xml file and go into the Design mode.

Underneath the file tabs you can choose between Design view and/or blueprint view.

Moving across to the right is a button that let's you change the orientation.

Then there's a drop down that let's you chose what device you preview, the API, app theme, language, and layout variants.

To the right of those you'll see zoom controls.

Below those you'll see a bunch of controls for Autoconnect. For now just make sure Autoconnect is on (it should be by default) by hovering over the magnet looking icon.

Now take a look in the palette to the left of the layout editor to see all the layouts and widgets available for use.

A text view has already been added for us. Click on it in the Component Tree and all the way to the right open up the Properties pane. Here you can see some of the attributes for the TextView. You can see more using the arrows or clicking on View all attributes.

Look at the XML for TextView by going into Text mode.

You can see all the same properties including the constraints that were automatically added.

String Resources

Notice the textview has `android:text="Hello World!"`

It's not good practice to hard code string values in your xml file, you should use string resources.

strings.xml in the values folder stores all the string resources.

Go into strings.xml and add this resource with some initial value so you can see it worked.

`<string name="text_message">Hello Android </string>`

Back in activity_main.xml change the textview to `android:text="@string/text_message"`

The '@' sign means it's defined as a resource.

Now go into Design mode or Preview and you will see our textview has the new string value.

You can click anywhere on `@string/text_message` and click cmd-B (ctrl B on a pc) to automatically open the strings.xml file where the values for the string resources are stored.

Let's also make the text larger by either scrolling in the Properties pane to textSize or adding in the xml

`android:textSize="48sp"`

We will always use the "sp" unit for font sizes as it allows the font size to scale for the user based on their settings. (scale-independent pixel)

sp has scalable ratio: $sp = px * ratio * scale$. Where ratio never changes, but scale is user configurable.

This will allow the font size to scale such as for people who set larger font sizes in their settings.

If this is a value you're going to be using throughout your layout you should create an entry in the dimens.xml file with a name so you can easily reuse it.

Now let's change the background color through the xml.

In the top level <ConstraintLayout tag before the close tag > add `android:background="#ff6232"`

Notice that a small colored square appears in the gutter to the left. Click on it and it brings up a color wheel. Open up the preview and you'll see the result.

Instant Run

To run your app again in the emulator you could click Run again, or you can use Instant Run.

Instant Run is the button with the lightening bolt to the right of the Run button and can quickly apply small changes to your app already running in the emulator without doing an entire new compile.

Therefore it's always faster, so I recommend using it when you can.

Code completion

The editor in Android Studio has code completion just like in Xcode. As you type you will get selections. To accept the top most suggestion, press the Enter or Tab key on the keyboard. To select a different suggestion, use the arrow keys to move up and down the list, once again using the Enter or Tab key to select the highlighted item.

In the XML file because everything starts with "android" it gets old typing that. You can just start typing what comes after the colon and the code suggestions will still come up. So instead of typing

`android:background` just start typing `background` and you'll see `android:background` as a choice.

To see other suggestions, click on a word and then use ctrl-space and the editor will show you a list of alternative suggestions.

Can you guess how you would change the font color for the textview?

In <TextView> add `android:textColor="#FFFFFF0D"`

Note: To open a project in Android Studio there is no one file you can click on to open the project (similar to the xcodeproj file). To open a project go into Android Studio and open an existing project from there. If you're opening a project that you did not create, such as my samples, chose import project instead to avoid configuration errors.

ATLS 4120: Mobile Application Development

Week 11: Android Design

App Design

The fundamentals of app design are the same regardless of platform.

1. Design for the device

- Mobile, mobile, mobile
- All apps should be easy to figure out and use
- Make onboarding quick and easy
 - Download, install, start instantly, no lengthy startup
 - Avoid requiring initial signup/login
 - Show content immediately
- Avoid unnecessary interruptions
- Interaction is through taps and gestures
 - The comfortable minimum size of tappable UI elements is 44 x 44 points
- No “Home” in apps
- Artwork and image should be useful and draw the user in
 - Adapt art to the screen size, high quality media is expected
- Handle different device sizes and orientations

2. Content

- Get users to the content they care about quickly
- Provide only relevant, appropriate content that’s useful to the immediate task
- If in doubt, leave it out

3. Focus on the User

- Know thy user - target apps to a specific user level
- Put the user in control
- Think through the user flow
- Get them to the relevant information quickly
- Provide subtle but clear, immediate feedback
- Create a compelling user experience
 - User interaction consistency

4. Focus on the task

- The goal of your app and the problem you’re trying to solve

5. Understand platform conventions

- Android launch screens historically have not been recommended
 - Material Design includes a launch screen pattern
<https://material.io/design/communication/launch-screen.html>
 - There is no launch screen ability built into Android Studio
 - Don’t use timers, it only delays the user and makes your app feel slow
 - Use a launcher theme or activity if you want a custom launch screen
<https://www.bignerdranch.com/blog/splash-screens-the-right-way/>
<https://android.jlelse.eu/the-complete-android-splash-screen-guide-c7db82bce565>

<https://plus.google.com/+IanLake/posts/SW7F2CJvAmU>

- Android devices have a back button, your apps don't need one
<https://developer.android.com/training/implementing-navigation/temporal>

Android Design

<https://developer.android.com/design/>

Material Design <https://material.io/>

Quality Guidelines <https://developer.android.com/develop/quality-guidelines/>

Material Design

Google launched Material design in 2014 to provide guidance and advice to developers to help them make the experience they create, better for their users.

Making Material Design

https://www.youtube.com/watch?v=rrT6v5sOwJg&list=PL8PWUWLnnIXPD3UjX931fFhn3_U5_2uZG

1. Material System <https://material.io/design/introduction/>
2. Material Foundation <https://material.io/design/foundation-overview/>
 - Color <https://material.io/design/color/the-color-system.html>
 - Palette generator color tool
<https://material.io/tools/color/#!/?view.left=0&view.right=0&primary.color=673AB7>
 - 2014 Material Design color palettes
 - Typography <https://material.io/design/typography/the-type-system.html#>
 - Iconography <https://material.io/design/iconography/product-icons.html>
 - Layout and pixel density <https://material.io/design/layout/understanding-layout.html#>
3. Material Guidelines <https://material.io/guidelines/>
 - Material Theme editor Sketch plugin
 - Icons <https://material.io/design/platform-guidance/android-icons.html#>

Material Design for Android

<https://developer.android.com/guide/topics/ui/look-and-feel/>

Android styles and themes let you separate the app design from the UI structure and behavior, similar to CSS in web design.

Styles and themes are declared in the style resource file `res/values/styles.xml`.

A style is a collection of attributes that specify the appearance for a single View.

- You can only apply one style to a View, but some views like `TextView` let you specify additional styles through attributes such as `textAppearance` on `TextViews`.
- You can extend an existing style from the framework or support library or your own project using the “parent” attribute.
- They are not inherited by child views

A theme is a type of style that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply your style as a theme, every view in the app or activity applies each style attribute that it supports.

To apply a theme use the `android:theme` attribute on either the `<application>` tag or an `<activity>` tag in the `AndroidManifest.xml` file.

Android lets you set styles attributes in different ways:

- directly in a layout
- programmatically
- apply a style to a view
- apply a theme to a layout

You should use themes and styles as much as possible for consistency. If you've specified the same attributes in multiple places, Android's style hierarchy determines which attributes are ultimately applied. The list is ordered from highest precedence to lowest:

1. Applying character- or paragraph-level styling via text spans to TextView-derived classes
2. Applying attributes programmatically
3. Applying individual attributes directly to a View
4. Applying a style to a View
5. Default styling
6. Applying a theme to a collection of Views, an activity, or your entire app
7. Applying certain View-specific styling, such as setting a TextAppearance on a TextView

Material Design Develop <https://material.io/develop/>

Material Design Tools <https://material.io/tools/>

- Theme editor plug-in for Sketch (requests for XD)
- Icons
- Color tool

Launcher Icons

https://developer.android.com/guide/practices/ui_guidelines/icon_design_adaptive

- Launcher(app) icons represent your app. They appear on the home screen, settings, sharing, and can also be used to represent shortcuts into your app
 - Legacy launcher icons Android 7.1(API 25 and before) 48x48dp
 - Adaptive icons were introduced in Android 8.0(API 26) which display as different shapes as needed. Each OEM provides a mask which is used to render the icons. You can control the look of your adaptive launcher icon by defining 2 layers, consisting of a background and a foreground.
 - Both layers must be sized at 108 x 108 dp.
 - The inner 72 x 72 dp of the icon appears within the masked viewport.
 - The system reserves the outer 18 dp on each of the 4 sides to create visual effects, such as parallax or pulsing.
 - We'll use the Image Asset Studio to create all our icons
- Android projects include default icons - `ic_launcher.png` and `ic_launcher_round.png`
- Launcher icons go into density specific `res/mipmap` folders (i.e. `res/mipmap-mdpi`)
- Launcher icons should be designed specifically for Android. Avoid mimicking visual elements and styles from other platforms.

- If you only include the a higher resolution version Android will generate the lower resolutions
- There are also required icons for the action bar, dialog & tab, notifications, and small contextual
- Tablets and other large screen devices request a launcher icon that is one density size larger than the device's actual density, so you should provide your launcher icon at the highest density possible.

Halloween (theme)

In the Android manifest file you'll see the activity app theme

`android:theme="@style/AppTheme"`

If you open styles.xml you'll see where the style name AppTheme is defined.

Let's look at how we can customize the theme and styles of our app.

When you create a project with Android Studio, it applies a material design theme to your app by default, as defined in your project's `styles.xml` file. This `AppTheme` style extends a theme from the support library and includes overrides for color attributes that are used by key UI elements.

```
<style name="AppTheme" parent="android:Theme.Material.Light.DarkActionBar">
</style>
```

If you checked backwards compatibility instead of Material you'll see `AppCompat`.

You can change the parent to a theme available for the API levels you supported.

Click on Open Editor to open the Theme Editor (or Tools | Theme Editor)

<https://developer.android.com/studio/write/theme-editor>

Look at the available themes

Back in `styles.xml` change the `AppTheme` style to use the parent

`android:ThemeOverlay.Material.Dark.ActionBar`

Run the app to see what it looks like. Try some other themes to see what they look like.

Open `colors.xml`

```
<color name="colorPrimary">#008577</color>
<color name="colorPrimaryDark">#00574B</color>
<color name="colorAccent">#D81B60</color>
```

You can customize your theme by adding or changing the colors in the `colors.xml` file.

Add a new color resource. When choosing a color pick the closest material color.

```
<color name="background">#ff9640</color>
```

We can use these by adding them in `styles.xml` in our `AppTheme` style.

```
<item name="android:colorPrimary">@color/colorPrimary</item>
<item name="android:colorPrimaryDark">@color/colorPrimaryDark</item>
<item name="android:colorAccent">@color/colorAccent</item>
<item name="android:background">@color/background</item>
```

We use `@color` because the style values are actually references to our color resources.

Creating a new theme

Back in the theme editor you can create a new theme.

Click on the theme select list, create new theme.

Theme name: AutumnTheme (no spaces allowed)

Parent theme name: android:Theme.Material.Light.DarkActionBar

Now we can change any of the colors we want.

You can change existing color resources by clicking next to the name of the resource you want to change and selecting a color.

The colors are listed in the left column of the Resources dialog and arranged into the following groups.

- Project: These are colors inside your project. Some can be edited because they are part of your project sources, and some cannot be edited because they are part of the libraries you have included in your project.
- android: These are color resources that belong to the android namespace. They are part of the Android framework and cannot be edited.
- Theme Attributes: These are attributes of the currently selected theme. They are referenced by the theme and can change depending on what theme you have selected. Theme attributes are never editable from inside the Resources dialog.

Or you can define new colors.

I'm going to use the color picker tool to generate a color palette <https://material.io/tools/color/>

Once you generate a palette you can export for Android and it will create an xml file for you.

(On Mac this opens in Xcode as default)

Copy these into colors.xml.

```
<color name="primaryColor">#ff8f00</color>
<color name="primaryLightColor">#ffc046</color>
<color name="primaryDarkColor">#c56000</color>
<color name="secondaryColor">#6d4c41</color>
<color name="secondaryLightColor">#9c786c</color>
<color name="secondaryDarkColor">#40241a</color>
<color name="primaryTextColor">#000000</color>
<color name="secondaryTextColor">#ffffff</color>
```

Use these in your new theme.

```
<style name="AutumnTheme" parent="android:Theme.Material.Light.DarkActionBar" >
  <item name="android:colorPrimary">@color/primaryColor</item>
  <item name="android:colorPrimaryDark">@color/primaryDarkColor</item>
  <item name="android:colorAccent">@color/secondaryLightColor</item>
  <item name="android:textColorPrimary">@color/secondaryDarkColor</item>
  <item name="android:textColor">@color/secondaryDarkColor</item>
  <item name="android:colorBackground">@color/primaryColor</item>
  <item name="android:colorButtonNormal">@color/primaryDarkColor</item>
</style>
```

To use this theme for your whole app update AndroidManifest.xml

```
android:theme="@style/AutumnTheme"
```


You can also define different styles for different Android versions if you want to use newer features while still being compatible with old versions. All you need is another styles.xml file saved in a values directory that includes the resource version qualifier.

res/values/styles.xml # themes for all versions
res/values-v21/styles.xml # themes for API level 21+ only

The common structure to do this is to define a base theme and then use it as the parent for any version specific themes so you don't need to duplicate any styles.

Launcher icons

<https://developer.android.com/studio/write/image-asset-studio.html>

Add your own launcher icon to your app. (pumpkin.png)

Select the res folder right-click and select New > Image Asset

Or File > New > Image Asset.

Icon type: Launcher Icons (Adaptive and Legacy)

Use Adaptive and Legacy if you're supporting Android 8, otherwise you can just do Legacy.

Name: leave as ic_launcher so you don't need to change it in the Android_manifest.xml file

Foreground Layer:

Layer Name: ic_launcher_foreground

- Select Image to specify the path for an image file.
- Select Clip Art to specify an image from the material design icon set.
- Select Text to specify a text string and select a font.

Scaling – trim and resize as needed.

Background Layer:

Layer Name: ic_launcher_background

- Asset Type, and then specify the asset in the field underneath. You can either select a color or specify an image to use as the background layer.

Scaling – trim and resize as needed.

Next

Res Directory: main

Output Directories: main/res

Finish

Image Asset Studio adds the images to the mipmap folders for the different densities.

Legacy only:

Asset Type: Image

Browse to your image.

Trim - To adjust the margin between the icon graphic and border in the source asset, select Yes. This operation removes transparent space, while preserving the aspect ratio. To leave the source asset unchanged, select No. Default: No

Padding - If you want to adjust the source asset padding on all four sides, move the slider. Select a value between -10% and 50%. If you also select Trim, the trimming happens first. Default: 0%

Foreground - To change the foreground color for a Clip Art or Text icon, click the field. In the Select Color dialog, specify a color and then click Choose. The new value appears in the field. Default: 000000

Background - To change the background color, click the field. In the Select Color dialog, specify a color and then click Choose. The new value appears in the field. Default: FFFFFFFF

Scaling - To fit the icon size, select Crop or Shrink to Fit. With crop, the image edges can be cut off, and with shrink, they aren't. You can adjust the padding, if needed, if the source asset still doesn't fit well. Default: Shrink to Fit

Shape - To place a backdrop behind your source asset, select a shape, one of circle, square, vertical rectangle, or horizontal rectangle. **For a transparent backdrop, select None.** Default: Square

Effect - If you want to add a dog-ear effect to the upper right of a square or rectangle shape, select DogEar. Otherwise, select None. Default: None

Image Asset Studio places the icon within a transparent square so there's some padding on the edges. The padding provides adequate space for the standard drop-shadow icon effect.

You will see the warning that an icon with the same name exists. That's because Android Studio provides default launcher icons, we're replacing those so just ignore this warning.

Next

Make sure it's saving your icons to src/main/res.

Finish

Now run your app and look at your launcher icon by clicking the right button, or go to home and then all apps to see it on your home screen.

You can see the various launcher files created in the mipmap folder.

You can download the Android Icon Templates Pack

https://developer.android.com/guide/practices/ui_guidelines/icon_design.html

You can also use Android Asset Studio to create all your launcher images.

<http://romannurik.github.io/AndroidAssetStudio/index.html>

ATLS 4120: Mobile Application Development

Week 11: Android Layouts

Layouts

A layout defines the visual structure for a user interface and acts as a container for your UI widgets.

<https://developer.android.com/guide/topics/ui/declaring-layout#CommonLayouts>

Before Android added constraint layouts developers used a combination of other layouts, often nesting them to achieve the layout they were aiming for.

- Linear layouts organize its widgets into a single horizontal or vertical row.
<https://developer.android.com/guide/topics/ui/layout/linear.html>
 - android:orientation determines which direction you want to arrange views in (required)
 - vertical: views are displayed in a single column
 - horizontal: views are displayed in a single row
- Relative layouts display views relative to each other or to their parent.
<https://developer.android.com/guide/topics/ui/layout/relative.html>

You can also build layouts dynamically using adapters. This is useful when your layout is not pre-determined and you want to populate it at run time. We'll get into adapters next semester.

- List Views display a scrolling list of data
- Grid views display items in a two-dimensional, scrollable grid.
<https://developer.android.com/guide/topics/ui/layout/gridview.html>

Constraint Layout

<https://developer.android.com/training/constraint-layout/index.html> 1:05 - end (4 mins)

Constraint Layout was added in Android Studio 2.2 to allow developers to create adaptive layouts. In constraint layouts views are laid out according to relationships between sibling views and the parent layout. Constraint layouts have a flat view hierarchy, so there's no nesting of other layouts. This is especially useful for large and complex layouts and helps with performance.

Constraint Layout is compatible with Android 2.3 (API level 9) and higher.

The new layout editor to create ConstraintLayout is available in Android Studio 2.2 and higher and became the default layout in Android Studio 3.0.

Overview


To define a view's position in ConstraintLayout there must be at least one vertical and one horizontal constraint for the view. Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.

Creating Constraints

There are 3 ways to create constraints: Auto-connect, Inference, and manually.

Auto-connect:


Autoconnect automatically creates two or more constraints for each view as you add them to the layout, but only when appropriate to constrain the view to the parent layout. Autoconnect does not create constraints to other views in the layout.

- You can enable Autoconnect by clicking Turn on Autoconnect in the Layout Editor toolbar . Auto connect is turned off by default.
- Autoconnect does not create constraints for existing views in the layout, only as you add views

Inference:


Infer Constraints is a one-time action that creates a set of constraints for either the view you have selected or all views in your layout if none are selected.

Manual:

If you disable auto connect or delete a views constraints you can create constraints manually. When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints. But the layout editor will show a missing constraint error , as the view will be placed at the top left if it has no constraints. Show Warnings and Errors 

Show Constraints (eye icon) in the toolbar will let you see the constraints in the design mode.

Constraint Rules

1. Every view must have at least two constraints: one horizontal and one vertical.
 - if a view has no constraints when you run your layout, it will be drawn at position [0,0] (the top-left corner).
 - A missing vertical constraint will cause it to be drawn at the top of the screen
 - A missing horizontal constraint will cause it to be drawn at the left of the screen
 - Missing a constraint won't cause a compile error
 - The Layout Editor indicates missing constraints as an error in the toolbar. To view the errors and other warnings, click Show Warnings and Errors 
2. You can create constraints only between a constraint handle and an anchor point that share the same plane. So a vertical plane (the left and right sides) of a view can be constrained only to another vertical plane; and baselines can constrain only to other baselines.
3. Each constraint handle can be used for just one constraint, but you can create multiple constraints (from different views) to the same anchor point.

Adding a Constraint

A constraint describes how a view should be positioned relative to other views in a layout as well as its parent. A constraint is typically defined for one or more sides by connecting the view to:

- An anchor point or another view
- An edge of the layout
- An invisible guide line

When you add a view in a ConstraintLayout, it displays a bounding box with square resizing handles on each corner and circular constraint handles on each side.

To create a constraint click the view to select it then click-and-hold one of the constraint handles. Drag the line to an anchor point (the edge of another view, the edge of the layout, or a guideline). When you release, the constraint is made, with a default margin separating the two views. You can adjust the margins by moving the view in the design preview or change the value in the properties window.

You can also add a constraint by clicking the + in the Attributes window view inspector area.

Constraint Types

Parent position:


Connect the side of a view to the corresponding edge of the layout.

Order position:

Define the order of appearance for two views, either vertically or horizontally.


Alignment:

Align the edge of a view to the same edge of another view.

You can align multiple views by selecting them and clicking Align  in the toolbar to select the alignment type.


Baseline alignment:

Align the text baseline of a view to the text baseline of another view.

- Select the textview you want to constrain and click Edit Baseline , which appears below the view. Then click the text baseline and drag the line to another baseline.

Constrain to a guideline:


You can add a vertical or horizontal guideline to which you can attach constraints. You can position the guideline within the layout based on either dp units or percent, relative to the layout's edge.


To create a guideline, click Guidelines  in the toolbar, and then click either Add Vertical Guideline or Add Horizontal Guideline. Drag the dotted line to reposition it and click the circle at the edge of the guideline to toggle the measurement mode.

Constrain to a barrier:

Barriers are a new addition to constraint layout. A barrier is an invisible line that you can constrain views to. A barrier does not define its own position, its position moves based on the position of views contained within it. This is useful when you want to constrain a view to a set of views rather than to one specific view.

To create a barrier:

1. Click Guidelines  in the toolbar, and then click Add Vertical Barrier or Add Horizontal Barrier.
2. In the Component Tree window, select the views you want inside the barrier and drag them into the barrier component.

3. Select the barrier from the Component Tree, open the Attributes  window, and then set the barrierDirection.


Now you can create a constraint from another view to the barrier.

You can also constrain views that are inside the barrier to the barrier. This way, you can ensure that all views in the barrier always align to each other, even if you don't know which view will be the longest or tallest.

Constraint Size Modes


To select from one of the dynamic sizing modes, click a view and open the Properties window and use the Inspector Pane to adjust the constraints. Click on the size mode symbols inside the square to toggle the size mode.

- android:layout_width and android:layout_height are required attributes for almost all widgets


 Fixed: You specify the dimension in the text box below or by resizing the view in the editor.

- android:layout_width and android:layout_height will have the fixed value

You can use the handles on each corner of the view to resize it, but doing so will set the width and height values to fixed values, which you should avoid for most views because hard-coded view sizes cannot adapt to different content and screen sizes.

 Wrap Content: The view expands only as much as needed to fit its contents.

- android:layout_width and android:layout_height will be “wrap_content”

 Match constraints: The view expands as much as possible to meet the constraints on each side after accounting for the views margins.

- android:layout_width and android:layout_height will be 0dp because the view has no desired dimensions, but it resizes as needed to meet the constraints.
- You can also set the constraint width to have a layout_constraintWidth_min or layout_constraintWidth_max.

You can't use match_parent for any view in a ConstraintLayout, use match constraints instead.

Adjusting constraints

Bias:

If you add constraints to both sides of a view (and the view size for the same dimension is either "fixed" or "wrap content"), the constraint lines become squiggly like a spring to indicate the opposing forces and the view becomes centered between the two constraints with a bias that's a percentage between 1(left or top) and 100(right or bottom). Centered, 50% is default. You can adjust the horizontal or vertical bias using the slider in the Properties window or by dragging the view. 1 and 100 are swapped in languages that are read right to left.

If you want the view to stretch its size to meet the constraints, switch the size to “match constraints”. In XML the percentages are represented as decimals from 0 to 1.

- Vertical bias allows you to position a view along the vertical axis using a bias value, this will be relative to its constrained position.
- Horizontal bias allows you to position a view along the horizontal axis using a bias value, this will be relative to its constrained position.

Size ratio:

You can set the view size to a width:height ratio if at least one of the view dimensions is set to "match constraints" (0dp). To set up a ratio click Toggle Aspect Ratio Constraint in the Properties window by clicking on the triangle in the top left corner of the square with the constraints (only visible if at least one of the view dimensions is set to match constraints)

View Margins:

To ensure that all your views are evenly spaced, click Margin **8** in the toolbar to select the default margin for each view that you add to the layout. The button changes to show your current margin selection. Any change you make to the default margin applies only to the views you add from then on.

You can control the margin for each view in the Properties window by clicking the number on the line that represents each constraint.

All margins offered by the tool are factors of 8dp to help your views align to Material Design's 8dp square grid recommendations.

Chains

A chain is a group of views that are linked to each other with bi-directional position constraints. A chain allows you to distribute a group of views horizontally or vertically.

To create a chain of views quickly, select them all, right-click one of the views, and then select either Center Horizontally or Center Vertically, to create either a horizontal or vertical chain.

Chain styles:

Once you've created a chain you can cycle through the chain styles by clicking on the chain icon on the views in the chain.

Spread


- The views are evenly distributed after margins are accounted for
- This is the default

Spread inside

- The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed.

Packed

- The views are packed together after margins are accounted for
- You can adjust the whole chain's bias by changing the chain's head view bias.

The chain's "head" view is the left-most view in a horizontal chain and the top-most view in a vertical chain, and defines the chain's style in XML. You can toggle between spread, spread inside, and packed by selecting any view in the chain and then clicking the chain button  that appears below the view.

Weighted

- When the chain is set to either spread or spread inside, you can fill the remaining space by setting one or more views to "match constraints" (0dp). By default, the space is evenly distributed between each view that's set to "match constraints," but you can assign a weight of importance to each view using the `layout_constraintHorizontal_weight` and `layout_constraintVertical_weight` attributes.
 - Weight is a proportion used to determine how views are distributed
 - For example if one view has a weight of 1 and a second view has a weight of 7, the first would get 1/8 the space, the second 7/8 of the space
 - the view with the highest weight value gets the most amount of space
 - views that have the same weight get the same amount of space.
 - If only 1 view has a weight of 1 it will get all the extra space

Chain rules:

- A view can be a part of both a horizontal and a vertical chain, which makes it easy to build flexible grid layouts.
- Each end of the chain must be constrained to another object on the same axis
- Using a vertical or horizontal chain does not align the views in that direction. Use other constraints to position each view in the chain, such as alignment constraints.

Deleting Constraints

You can delete a single constraint by clicking on the side of the box in the properties window or in the design view on the handle of the constraint that you wish to delete.

When a view is selected, a little icon will appear to the bottom left hand side - clicking this icon will remove **all** of the constraints that have been set on that view.

In the toolbar Clear All Constraints will clear the constraints on all views in your layout.

Attributes

<https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html>

Halloween

Let's go through all the constraints our views currently have and understand them.

All have `android:layout_width` and `android:layout_height` set to `wrap_content`. Toggle them to `match_constraints` to see how they change. Note that you can see the triangle for size ratio appear and if you click it you can set the ratio. You should avoid fixed values.

Notice that if you drag a view it will update the margins and bias according to its new position.

Adding a View

Make a little room above the edittext to add a textview.

Turn Autoconnect off to see what happens when I add a textview.

Since autoconnect was off no constraints were added so even though it looked right in design mode, it will automatically be placed at (0,0)

We could use Infer Constraints but instead let's look at how we can create the constraints manually.

Manually Creating Constraints

Click on the text view and you'll see that when a view is added to a ConstraintLayout it displays a bounding box with circular constraint handles on each side and square resizing handles on each corner.

Click-and-drag the top constraint handle to an available anchor point (the edge of another view, the edge of the layout, or a guideline) such as the top of the layout. When you release, the constraint is made, with a default margin that you can change in the Attributes pane.

You can also create constraints in the Attributes pane using the +. Wherever there's a + it means you could add a constraint. It will add it with the margin the view currently has in the layout.

If you click the + for a bottom constraint it will create one to the view below it, the edit text. If wanted it to the bottom layout you can change it to be to "parent" and then either change the margin or the vertical bias. Or create it by dragging the constraint handle to the bottom layout and it will create a bottom constraint to parent. But you don't need a bottom constraint because the top one handles the vertical placement.

Click and drag the left circle to the left margin and then release. Do the same thing with the circle on the right. You should now see it added two constraints. Since it is now horizontally and vertically positioned the constraint error went away and if you run it the text view is now centered. You can move the slider to change the bias.

You can also use Align in the toolbar and chose horizontal.

So in a constraint layout you either should have Autoconnect on, use Infer Constraints, or create constraints manually.

Since we're not going to keep this textview, let's look at how you can delete its constraints. I can hover over a single constraint in the Attributes pane or in the Design model and click to delete a single constraint.

When you select a view you will see an option at its bottom left corner to Delete Constraints. This will delete all the constraints for that view.

In the toolbar you can chose Clear All Constraints and that will clear the constraints on ALL the views in your layout. (Luckily you can undo this if you do it by mistake)

Practice creating constraints manually by clearing all the constraints and then creating them manually.

Landscape

If you run this app and rotate to landscape you'll see it doesn't look to good.

(Note: When you rotate it seems to "reset". It is working properly. When rotation occurs on Android the activity is killed and restarted. We'll talk about why next week when we discuss the Android lifecycle).

Android categorizes device screens using two general properties: size and density. The screens orientation (landscape or portrait) is considered a variation of screen size. Android automatically scales your layout in order to properly fit the screen. So your layouts for different screen sizes don't need to worry about the absolute size of UI elements but instead focus on the layout

structure that affects the user experience (such as the size or position of important views relative to sibling views). If you can't define one layout that works on all of these you need to define different layouts for each screen size.

Let's see if we can change the constraints so that this layout can work in both portrait and landscape.

Starting from the top I'm going to update the edittext to have a top constraint instead of a bottom constraint so I'm not wasting space at the top.

I deleted the bottom constraint and created a top constraint to the parent.

I did the same for the button, delete the bottom constraint and added a top constraint to the edittext.

Same for the textview, delete the bottom constraint. I also deleted/changed the top constraint to go to the bottom of the button (instead of the parent). It might be helpful to put in some temporary text while you're working on the layout.

The imageview required the most work as I needed the image to scale. First I deleted/changed the top constraint to go to the bottom of the textview(id message). I also updated the bottom constraint to be 8dp.

Then I changed all the imageview constraints to match_constraint so that I could set up a ratio.

Once you change all 4 constraints to match_constraint make sure they have

android:layout_width="0dp" and **android:layout_height="0dp"**.

Then I clicked on the triangle in the top left of the square in the Attributes pane and set up a 1:1 ratio. This will allow the image to scale, keeping its aspect ratio, to match the constraints.

For this layout I standardized on a 8dp margin to the parent and 16dp margin between the views in the layout.

Run it and now see what it looks like in landscape.

Previous layout values:

```
<EditText
    android:id="@+id/editText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="40dp"
    android:ems="10"
    android:hint="@string/name"
    android:inputType="textPersonName"
    app:layout_constraintBottom_toTopOf="@+id/button"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="40dp"
    android:onClick="sayBoo"
    android:text="@string/boo"
    app:layout_constraintBottom_toTopOf="@+id/message"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

<TextView
    android:id="@+id/message"
    android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="25dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

```

New layout values:

```

<EditText
    android:id="@+id/editText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:ems="10"
    android:hint="@string/name"
    android:inputType="textPersonName"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:onClick="sayBoo"
    android:text="@string/boo"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/editText" />

<TextView
    android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:textSize="24sp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/button" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginBottom="8dp"
    android:layout_marginTop="16dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintDimensionRatio="1:1"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/message" />

```

ATLS 4120: Mobile Application Development

Week 11: Android UI Widgets

Widgets

The Android SDK comes with many widgets to build your user interface. We've already looked at text views, edit texts, buttons, and image views. Today we're going to look at 5 more.

<https://developer.android.com/guide/topics/ui/controls.html>

Selection controls <https://material.io/guidelines/components/selection-controls.html#selection-controls-switch>

Toggle Button

<https://developer.android.com/guide/topics/ui/controls/togglebutton.html>

- A toggle button allows the user to choose between two states **<ToggleButton .../>**
<https://developer.android.com/reference/android/widget/ToggleButton.html>
- The **android:textOn** attribute determines the text on the button when the state is ON
- The **android:textOff** attribute determines the text on the button when the state is OFF
- The **isChecked()** method returns a boolean – true if it's on, false if it's off
- Available since API 1

Switch

- A switch is a two state toggle switch that slides **<Switch .../>**
<https://developer.android.com/reference/android/widget/Switch.html>
- The **android:textOn** and **android:textOff** attributes determine the text you want to display depending on the state of the switch
- The **isChecked()** method returns a boolean – true if it's on, false if it's off
- Added in API 14

CheckBox

<https://developer.android.com/guide/topics/ui/controls/checkbox.html>

- Check boxes let you display multiple options that the user can check or not **<CheckBox .../>**
<https://developer.android.com/reference/android/widget/CheckBox.html>
- Each check box is independent of the others
- The **isChecked()** method returns a boolean – true if it's checked, false if it's not

RadioButton

<https://developer.android.com/guide/topics/ui/controls/radiobutton.html>

- RadioGroup is a container for radio buttons **<RadioGroup .../>**
- Radio buttons let you display multiple options **<RadioButton .../>**
<https://developer.android.com/reference/android/widget/RadioButton.html>
- Users can select only ONE radio button in a radio group
- The **getCheckedRadioButtonId()** method returns the id(integer) of the chosen button
 - -1 means no button was chosen

Spinner

<https://developer.android.com/guide/topics/ui/controls/spinner.html>

- A spinner presents a drop-down list of values from which only one can be selected **<Spinner .../>** <https://developer.android.com/reference/android/widget/Spinner.html>
- You can store the values as an array in strings.xml

- The **getSelectedItem()** method returns the String of the selected item

Properties

Widgets that are descendants from the View class have some commonly used properties available

- **android:id**
 - Gives the component a unique identifying name
 - Lets you access the widget in your code
 - Lets you refer to the widget in your layout
- **android:text**
 - the text displayed in that component
- All controls will have **layout_width** and **layout_height**

Toasts

<https://developer.android.com/guide/topics/ui/notifiers/toasts.html>

A toast is a small popup that can be used to provide simple feedback. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

Sport

From the Welcome screen chose Start a new Android Studio Project (File | New | New Project)

Application Name: Sport

Company name: a qualifier that will be appended to the package name

Project location: the directory for your project

Package name: the fully qualified name for the project

Leave C++ and Kotlin unchecked

Form factors: Phone and Tablet (leave others unchecked)

Minimum SDK: API 21 (Android 5.0 Lollipop)

Add an Activity to Mobile: Empty Activity

Activity Name: MainActivity (we can leave this)

Make sure Generate Layout File is checked

Layout name: activity_main

Backwards Compatibility should not be checked

Open the activity_main.xml file.

In Design mode move the textview to the top. We'll use this as a heading so I changed the textAppearance to be **Material.Headline**

Add a toggle button below it. Notice it was added with the id "toggleButton".

In Design mode add a button toward the bottom of the layout. It was added with the id "button".

Add a textview below the button, all the way at the bottom, we'll use that for the result.

Give it an id so we can refer to it in our code.

android:id="@+id/sportTextView"

We don't want it to have any default text, so remove that. But I do want the text a little larger and centered.

android:textAlignment="center"

android:textAppearance="@android:style/TextAppearance.Material.Medium"

These all got created with default text, let's change that.

In strings.xml add string resources.

```
<string name="heading">Find Your Sport</string>
<string name="sport_button">Find</string>
<string name="toggle_on">inside</string>
<string name="toggle_off">outside</string>
```

TextView

In activity_main.xml update the top textView to use the string resource

```
android:text="@string/heading"
```

Toggle Button

In activity_main.xml update the toggleButton to use the string resource by replace the text property with

```
android:textOn="@string/toggle_on"
android:textOff="@string/toggle_off"
```

Button

In activity_main.xml update the button to use the string resource

```
android:text="@string/sport_button"
```

Now we want the button to do something. In the button tag start typing onclick and use autocomplete.

```
android:onClick="findSport"
```

findSport is the method we want the button to call when it's clicked.

Either click on the lightbulb and chose Create findSport(View) in MainActivity or just go into MainActivity.java and create it.

Android looks for a public method with a void return value, with a method name that matches the method specified in the layout XML.

The parameter refers to the GUI component that triggers the method (in this case, the button).

```
public void findSport(View view){
```

```
ToggleButton toggle = findViewById(R.id. toggleButton);
```

```
    boolean location = toggle.isChecked();
```

```
    String perfectSport;
```

```
    if(location){
```

```
        perfectSport="Yoga";
```

```
    }
```

```
    else{
```

```
        perfectSport="Running";
```

```
    }
```

```
    TextView sportSelection = findViewById(R.id.sportTextView);
```

```
    sportSelection.setText(perfectSport + " is the sport for you");
```

```
}
```

We create a toggle object so we have a reference to our ToggleButton.

The findViewById() method is how Java can get access to the UI components.

The R.java class is automatically generated for us and keeps track of all our IDs.

R.id.toggleButton grabs a reference to our toggle button. (note that R must be capitalized).

The toggle button's `isChecked()` method returns true or false and we store that in a Boolean variable. Then we create a string called `perfectSport` that will eventually store the sport that's picked. We use an if/else statement so if the toggle button is on(inside) then `perfectSport` is assigned yoga. If it's not on(outside) then `perfectSport` is assigned running. We create a `TextView` object called `sportSelection` and make it a reference to our `sportTextView` `textView`.

We can set the text by using the `setText()` method.

Remember your semi colons!

Expand your import statements and make sure they're either being added automatically or you're adding an import line for each new class we use. Auto import might not work if you're pasting code into your java file, but you shouldn't be doing that anyway!

Now before we run it we have some constraint errors to take care of. You'll notice that the toggle button and button are complaining of not having a vertical constraint. In design mode select one and chose Infer Constraints(magic wand), and then do the other. Those errors should go away and not you can run your app and make sure it works.

Spinner

Add a spinner to the right of the toggle button. Notice it's been given the id "spinner".

We'll store the values for the spinner in an array.

Just like we defined strings we can define a string-array in `strings.xml`. This works when the items in the array won't change. We'll deal with changing lists next semester.

```
<string-array name="exercise">
    <item>cardio</item>
    <item>strength</item>
    <item>flexibility</item>
</string-array>
```

In `activity_main.xml` let's reference this resource. Start typing entries and use auto complete.

```
android:entries="@array/exercise"
```

Now let's add logic to our Java to use the exercise type. Update `findSport()`

```
Spinner exercise = findViewById(R.id.spinner);
String exerciseType = String.valueOf(exercise.getSelectedItem());
```

```
if(location){ //inside
    switch (exerciseType){
        case "cardio":
            perfectSport="Spin class";
            break;
        case "strength":
            perfectSport="Weight training";
            break;
        case "flexibility":
            perfectSport="Yoga";
            break;
        default:
            perfectSport="Yoga";
    }
}
```

```

}
else { //outside
    switch (exerciseType){
        case "cardio":
            perfectSport="Running";
            break;
        case "strength":
            perfectSport="Climbing";
            break;
        case "flexibility":
            perfectSport="Yoga";
            break;
        default:
            perfectSport="Yoga";
    }
}

```

We create an exercise object to reference our spinner.

Then we create a string and get the value of the selected item in the spinner.

We use this string in a switch statement in our if/else statement.

In Java switch statements do fall through so you need a break after each case. A default is also required.

Fix the spinner's missing constraint using infer constraints and use instant run to update the emulator.

Radio Buttons

Next we're going to add radio buttons to choose cost of the sport.

Radio buttons belong in a group so add a radio button group first and make it the full width of the layout. Make the orientation horizontal.

android:orientation="horizontal"

Notice it has the id radioButton.

It will need horizontal and vertical constraints if it doesn't have any.

Add 3 radio buttons into the radio group going across in a row.

Notice they're each added with layout_weight of 1. This will make them be equally spaced across the group.

Update their ids to be radioButton1, radioButton2, and radioButton3.

In strings.xml add

```

<string name="radio1">${}</string>
<string name="radio2">${}</string>
<string name="radio3">${}</string>

```

Then update the radio button's text property to use these strings.

Add logic to MainActivity.java

```

RadioGroup cost = findViewById(R.id.radioButton);
int cost_id = cost.getCheckedRadioButtonId();

```

getCheckedRadioButtonId() returns the id of the radio button picked in the radio group.

-1 is returned if no button is selected.

Let's also use a Toast popup to notify the user if they didn't chose any cost level. Toasts are an easy way to provide a notification if no user response is needed.

<https://developer.android.com/guide/topics/ui/notifiers/toasts.html>

In Android the Context class provides access to the app environment which allows the launching of application-level operations. `getApplicationContext()` returns the app's context.

```
if (cost_id == -1) {  
    //toast  
    Context context = getApplicationContext();  
    CharSequence text = "Please select a cost level";  
    int duration = Toast.LENGTH_SHORT;  
    Toast toast = Toast.makeText(context, text, duration);  
    toast.show();  
} else {  
    if (location) { //inside  
        if (cost_id == R.id.radioButton1) { //cheapest option  
            perfectSport = "A home workout";  
        } else {  
            switch (exerciseType) {  
                case "cardio":  
                    perfectSport = "Spin class";  
                    break;  
                case "strength":  
                    perfectSport = "Weight training";  
                    break;  
                case "flexibility":  
                    perfectSport = "Yoga";  
                    break;  
                default:  
                    perfectSport = "Yoga";  
            }  
        }  
    }  
}
```

Check boxes

Let's add 4 checkboxes in a row. Give them ids checkBox1- checkBox4

Select them all, right click Chain | Create horizontal chain. I set the chain mode to spread by cycling through the chain modes.

The left most checkbox will need a constraint to the left, and the right most checkbox a constraint to the right.

Then I selected them all again, right click Align | Top Edges.

Then add a vertical constraint for one of them and all constraint conditions should be satisfied.

Add string resources

```
<string name="winter">Winter</string>  
<string name="spring">Spring</string>  
<string name="summer">Summer</string>  
<string name="fall">Fall</string>
```

Update activity_main.xml to use these 4 strings for the checkbox's text attribute.

Add logic to MainActivity.java

```
CheckBox winterCheckBox = findViewById(R.id.checkBox1);  
Boolean winter = winterCheckBox.isChecked();
```

```
CheckBox springCheckBox = findViewById(R.id.checkBox2);  
Boolean spring = springCheckBox.isChecked();
```

```
CheckBox summerCheckBox = findViewById(R.id.checkBox3);  
Boolean summer = summerCheckBox.isChecked();
```

```
CheckBox fallCheckBox = findViewById(R.id.checkBox4);  
Boolean fall = fallCheckBox.isChecked();
```

```
else { //outside  
    if(cost_id == R.id.radioButton3){  
        if(winter) {  
            perfectSport = "Skiing";  
        } else {  
            perfectSport = "Sky Diving";  
        }  
    } else {  
        switch (exerciseType) {  
            case "cardio":  
                perfectSport = "Running";  
                break;  
            case "strength":  
                perfectSport = "Climbing";  
                break;  
            case "flexibility":  
                perfectSport = "Yoga";  
                break;  
            default:  
                perfectSport = "Yoga";  
        }  
    }  
}
```

You have to make sure that all paths lead to perfectSport having a value or Java will give you an error. Or you can initialize it with a string to start with.

ATLS 4120: Mobile Application Development

Week 11: Java

History

- Java was created by James Gosling, Mike Sheridan, and Patrick Naughton
- Sun Microsystems released Java 1.0 in 1995
- In 2006 and 2007 Sun released Java as free and open-source software, under the terms of the GNU General Public License (GPL)
- Sun was bought by Oracle in 2009/2010, and Oracle continues to support Java

Java

- Java is a fully object-oriented programming language used by many software developers
 - Java has many standard libraries
 - Android provides a subset of these only for Android
- Java is a compiled language that is platform independent
 - The compiler creates bytecode that the Java Virtual Machine(JVM) then interprets into machine language
 - Bytecode is similar to machine language but not associated to any specific CPU
 - Android uses Ahead of Time(AoT) compilation to convert the bytecode into optimized machine code using the Android Runtime(ART) virtual machine when installed on an Android device to improve run time performance
 - *AoT* compilation only occurs in Android KitKat (4.4) and above, but it also provides backwards compatibility. Prior versions of Android relied on another virtual machine known as *Dalvik*.
 - Processing is also build on top of Java

Data Types

- Java has standard primitive data types
 - int
 - float
 - double
 - char
 - boolean
- Java is a strongly typed language so you need to specify the data type when declaring a variable

Variables and Constants

Java is case sensitive and all the naming conventions you're used to apply.

- Start with a letter, \$, or _ (not a number)
- Avoid key words
- Constants use the keyword **final** and are usually capitalized
 - Constants can't have their value changed
 - A final method means you can't override it
 - A final class means you can't create a subclass

<https://repl.it/@aileenjp/Java-intro>

```
double temp;  
temp=50;  
System.out.println(temp);
```

```
final int FREEZING = 32;
System.out.println(FREEZING);
```

```
FREEZING = 30; //error, can't change the value of a constant
```

If else

In Java the test condition must be in parenthesis. The body must be in curly brackets if there's more than one instruction.

- Java has all the typical logical and relational operators (==, !=, <, <=, >, >=, &&, ||)
- Java also supports the ternary operator shorthand
 - (expression) ? value if true : value if false

String equality

- Don't use == to test for String value equality, use **.equals()** instead
- == tests for reference equality (compares memory addresses to see if they are the same object)
- **.equals()** tests the string contents for value equality (whether they are logically equivalent).

```
if (temp > FREEZING){
    System.out.println("It's nice out");
} else {
    System.out.println("It's cold out");
}
```

Change temp to 30 to test the else statement.

Switch

A switch statement compares a variable's value to a series of case values and when a match is found the statements in that case are run.

- The variable used in a switch statement can only be integers, convertible integers (byte, short, char), strings and enums.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- Each case can be terminated with a break statement which terminates the switch statement
- If a break statement is not supplied, the code will continue executing into the next case statement until a break is reached
- You can also supply a default option to execute if none other cases apply

```
String grade = "B";
switch (grade)
{
    case "A":
        System.out.println("Excellent!");
        break;
    case "B":
        System.out.println("Well done");
        break;
    case "C":
```

```

case "D":
    System.out.println("You passed");
    break;
case "F":
    System.out.println("Try again");
    break;
default:
    System.out.println("Invalid grade");
    break;
}

```

Change grade to “C”. Note that case C has no instructions and no break so even if this case is a match it will go on to D and run the instructions in that case. This is called fall through and is fine if done intentionally.

Arrays

In Java arrays are a fixed size collection of elements all of the same data type.

- Arrays can contain both primitive data types as well as objects of a class
- array members are accessed using []

```
char[] sizes = {'s', 'm', 'l'};
```

```

String[] more_sizes = new String[4];
more_sizes[0] = "extra small";
more_sizes[1] = "small";
more_sizes[2] = "medium";
more_sizes[3] = "large";

```

If you try to add an item to an array that would go over its size you will get an array index out of bounds exception. `java.lang.ArrayIndexOutOfBoundsException`

```
more_sizes[4] = "extra large";
```

ArrayList is a dynamic sized arrays in Java that implement List interface. ArrayList is part of collection framework in Java.

- ArrayList only supports object entries, not the primitive data types.
- ArrayList has a set of methods to access elements and modify them, can't use []

(add at the very top of the repl)

```
import java.util.ArrayList;
```

```

ArrayList<String> size_list = new ArrayList<>();
size_list.add("small");
System.out.println(size_list);

```


You can add as many items as you want to an ArrayList

```
size_list.add("medium");  
System.out.println(size_list);
```

There are other data structures and collection types in Java that we'll get to next semester.

Loops

Java supports the common loops, for, while, and do while as well as the for each loop.

```
for(int i=0; i<size.length; i++){  
    System.out.println(size[i]);  
}
```

```
for(String mysize: more_sizes){  
    System.out.println(mysize);  
}
```

```
for(String mysize: size_list){  
    System.out.println(mysize);  
}
```

OOP

Java is an object-oriented programming language.

Classes

- A class is a template/blueprint that describes the behavior/state that the object of its type supports.
 - Data members for state (variables)
 - Method members for behavior (methods)
- In Java it's an Abstract Data Type as it describes a real-world abstraction
- You can control the visibility of a class as well as its variables and methods by specifying the access level
- The access levels are
 - public: accessible outside the class
 - private: only accessible inside the class
 - protected: accessible by the class and its subclasses
- Variables should be private and be accessible only by the method members declared for the same class
 - No methods outside a given class can access the private data members of the class.
- Class names should start with a upper case letter

Methods

A method is where the logic is written to define a specific behavior. Data is manipulated and all the actions are executed.

- Public methods can be accessed from outside the class and provide the public interface to the class
- Private methods are helper methods used inside the class
- Method names should start with a lower case letter

- Getter methods are used to get data
- Setter methods are used to set data

Java uses pass by value (not reference) for everything.

```
class Animal {
    //instance variables
    private int weight;
    private String name;

    //methods
    //setter
    public void setName(String animalName){
        name = animalName;
    }
    //getter
    public String getName(){
        return name;
    }
}
```

Objects

An object is an instance, or occurrence, of a given class. An object of a given class has the structure and behavior defined by the class that is common to all objects of the same class.

Many different objects can be defined for a given class with each object made up of the data and methods defined by the class

It's only when an object is instantiated that memory is allocated

Objects are instantiated by calling the class's constructor method

In Java, objects store the reference to the memory where its data is stored

```
Animal animal1 = new Animal();
animal1.setName("Fred");
System.out.println(animal1.getName());
```

Constructors

Constructor methods are initializer methods

Classes have constructor methods to initialize objects of that class

- Constructors have the same name as the class
- A class can have multiple constructors that each take a different number and/or type of parameters
- Constructors don't have a return type
- If you don't provide a constructor method Java will create one

```
public Animal(){
}
```

```
public Animal(String animalName){
    name = animalName;
}
```

```
public Animal(int animalWeight, String animalName){
    weight = animalWeight;
    name = animalName;
}
```

```
Animal animal2 = new Animal("Wilma");
Animal animal3 = new Animal(20, "BamBam");
System.out.println(animal2.getName());
System.out.println(animal3.getName());
```

Variable types

There are three kinds of variables in Java –

- Local Variables
 - Local variables are declared in methods, constructors, or blocks.
 - Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
 - Their scope is where they are declared -- method, constructor, or block.
 - There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.
 - Access modifiers cannot be used for local variables.
- Instance Variables
 - Instance variables are declared in a class, but outside a method, constructor or any block as they store essential data for an object's state.
 - The instance variables are visible for all methods, constructors and block in the class and can be accessed directly by calling the variable name inside the class.
 - Instance variables are created, and space allocated, when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
 - Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
 - Access modifiers can be given for instance variables.
- Class/Static Variables
 - Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
 - There is only one copy of each class variable per class, regardless of how many objects are created from it.
 - Static variables are usually used for declaring constants.
 - Static variables are created when the program starts and destroyed when the program stops.
 - Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.

- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor.
- Static variables can be accessed by calling with the class name
ClassName.VariableName.

Encapsulation

Encapsulation means data and operations are packaged into a single well-defined programming unit
In Java, the class provides for encapsulation

- The getter and setter methods provide the public class interface that is accessible from outside the class
- The private data members provide the information that is accessible only from within the class, thus providing the information hiding

We've already used some getter and setter methods

getText() EditText

setText() TextView

Inheritance

Inheritance allows you to define a hierarchy of classes with subclasses acquiring the properties of the superclass.

```
class Dog extends Animal {
    private String breed;

    public void setBreed(String dogBreed){
        breed = dogBreed;
    }

    public String getBreed(){
        return breed;
    }
}
```

```
Dog pet = new Dog();
pet.setName("Marmaduke");
pet.setBreed("Great Dane");
System.out.println(pet.getName() + " is a " + pet.getBreed());
```

If you tried to call getBreed() on an object of the Animal class you'd get an error.

```
animal1.getBreed();
```

Casting

Casting is taking an object of a certain class and changing its type to be of a different class

- Downcasting turns an object into a more specific type of object so you can have access to the methods and properties of that class
- Upcasting turns an object into a more generic type of object
- You will get a ClassCastException error if it's an illegal cast

- Downcasting is the more common direction for casting

Java String class

- The String class is immutable, so that once a String object is created cannot be changed.
- When you assign it a new value it's actually creating a new object.

<http://developer.android.com/reference/packages.html>

Java Packages

A package groups related classes together in one directory whose name is the same as the package name

- The java.lang package provides a number of helpful classes such as the String class
- The android.app package contains high-level classes encapsulating the overall Android application model

Comments

- Java has the standard comments
 - // single line
 - /* multi-line */

Interfaces

In Java an interface can be defined as a contract between objects on how to communicate with each other.

An interface defines the methods, a deriving class (subclass) should use. But the implementation of the methods is totally up to the subclass.

A subclass that implements an interface must implement the required methods and has the choice to implement any optional ones.

Debugging

Debug messages

- To log messages to the console in Java use **System.out.println("string");**
- Print messages to the LogCat
 - DEBUG: Log.d(tag, message);
 - ERROR: Log.e(tag, message);
 - INFO: Log.i(tag, message);
 - VERBOSE: Log.v(tag, message);
 - WARN: Log.w(tag, message);
- Log.i("TAG", "in start of count method");
- Filter to just see the logs for your tags

Debugging <https://developer.android.com/studio/debug/index.html>

More info on Logcat <https://developer.android.com/studio/debug/am-logcat.html>

ATLS 4120: Mobile Application Development

Week 12: Activities and Intents

Android follows the model view controller (MVC) architecture

- Model: holds the data and classes
- View: all items for the user interface
- Controller: links the model and the view together. The backbone or brain of the app.
- These categories should never overlap.

Activities

<https://developer.android.com/guide/components/activities/intro-activities.html>

<https://developer.android.com/reference/android/app/Activity.html>

- An activity is a single, specific task a user can do
- Each activity has its own window for its view The window typically fills the screen, but may be smaller than the screen and float on top of other windows
- An app can have as many activities as needed
- Each activity is listed in the AndroidManifest.xml file
- There is typically a 1:1 ration for activity and layout

To start an activity you need to define an intent and then use the startActivity(Intent) method to start a new activity.

Intents

<https://developer.android.com/guide/components/intents-filters.html>

<https://developer.android.com/reference/android/content/Intent.html>

Intents request an action such as starting a new activity.

- Provides the binding between two activities

You build an Intent with two key pieces of information

- Action – what action should be performed
- Data – what data is involved

There are two types of intents

- An explicit intent tells the app to start a specific activity
 - Provide the class name
- An implicit intent declares what type of action you want to perform which allows a component from another app to handle it
 - the app will start any activity that can handle the action specified
 - Android uses intent resolution to see what activities can handle the intent
 - An intent filter specifies what types and categories of intents each component can receive
 - Each activity has intent filters defined in the AndroidManifest.xml file
 - Android compares the information given in the intent with the information given in the intent filters specified in the AndroidManifest.xml file.
 - An intent filter must include a category of android.intent.category.DEFAULT in order to receive implicit intents
 - If an activity has no intent filter, or it doesn't include a category name of android.intent.category.DEFAULT, it means that the activity can't be started with an implicit intent. It can only be started with an explicit intent using the fully qualified component name.

- Android first considers intent filters that include a category of `android.intent.category.DEFAULT`
- Android then matches the action and mime type with the intent filters
- If just one activity can handle the intent, that activity is chosen
- If there is more than one activity that can handle the intent, the user is presented a list to choose from
- Android tells the activity to start even though it's in another app and passes it the intent

Data

<https://developer.android.com/training/basics/firstapp/starting-activity.html>

You can add extra information to your intent to pass data to the new activity using the `putExtra(String, value)` method

- The `putExtra(String, value)` method is overloaded so you can pass many possible types
- Call `putExtra(String, value)` as many times as needed for the data you're passing
- Each call to `putExtra(String, value)` is setting up a key/value pair and you will use that key to access that value in the intent you're starting.

When a new activity starts it needs to receive any data passed to it in the intent using the `getStringExtra()` methods.

Using intents Android knows the sequence in which activities are started. This means that when you click on the Back button on your device, Android knows exactly where to take you back to.

Events

<https://developer.android.com/guide/topics/ui/ui-events.html>

Android enables you to easily respond to common events through event listeners on the View class.

An event listener is an interface in the View class that contains a single callback method. These methods will be called automatically by the Android framework when the View to which the listener has been registered is triggered by user interaction with the UI control.

- Implement the listener
- Implement the callback method
- Assign the interface to a UI control

Coffee

Create a new project called Coffee

Minimum SDK: API 21

Empty Activity template

Activity name: FindCoffeeActivity

Check Generate Layout File

Layout Name: activity_find_coffee

Don't check Backwards Compatibility

User Interface

Use the textview provided as a heading that says "Coffee Shop Finder".

Change text appearance to be Material.Title

Also change its text property to use a string resource.

Make sure AutoConnect is on.

Add a spinner and a textview above it that will describe the spinner with text "Choose your crowd"

Add a button with the text "Find Coffee".

Add an imageView below the button. Copy an image into the res/drawable folder and use that as the src in the xml. You should also add a contentDescription using a string resource. The spinner, button, and image all should have an id since we'll be referring to these from our code. If you add these from top to bottom they will be added to the layout below each other. Make sure you use string resources for all text.

Strings.xml

```
<resources>
    <string name="app_name">Coffee</string>
    <string name="title">Coffee Shop Finder</string>
    <string name="coffee_type">Choose your crowd</string>
    <string name="button">Find Coffee</string>
    <string-array name="crowd">
        <item>popular</item>
        <item>cycling</item>
        <item>hipster</item>
        <item>tea</item>
        <item>hippie</item>
    </string-array>
    <string name="coffee_image">coffee cup</string>
</resources>
```

Java class

We're going to create a custom Java class for coffee shop info.
In the app/java folder select the coffee folder (not androidTest or test)
File | New | Java class (or right click)
Select .../app/src/main/java
Name: CoffeeShop
Kind: Class

We're going to create a CoffeeShop class with two data members to store the coffee shop name and URL.

Create getter and setter methods for both. We'll also create a private utility method that chooses the coffee shop so both setter methods can call this method instead of duplicating the functionality.

```
public class CoffeeShop {
    private String coffeeShop;
    private String coffeeShopURL;

    private void setCoffeeInfo(Integer coffeeCrowd){
        switch (coffeeCrowd){
            case 0: //popular
                coffeeShop="Starbucks";
                coffeeShopURL="https://www.starbucks.com";
                break;
            case 1: //cycling
                coffeeShop="Amante";
                coffeeShopURL="http://www.amantecoffee.com/";
                break;
        }
    }
}
```

```

    case 2: //hipster
        coffeeShop="Ozo";
        coffeeShopURL="https://ozocoffee.com";
        break;
    case 3: //tea
        coffeeShop="Pekoe";
        coffeeShopURL="http://www.pekoeshouse.com";
        break;
    case 4: //hippie
        coffeeShop="Trident";
        coffeeShopURL="http://www.tridentcafe.com";
        break;
    default:
        coffeeShop="none";
        coffeeShopURL="https://www.google.com/search?q=boulder+coffee+shops&ie=utf-8&oe=utf-8";
    }
}

```

```

public void setCoffeeShop(Integer coffeeCrowd){
    setCoffeeInfo(coffeeCrowd);
}

```

```

public void setCoffeeShopURL(Integer coffeeCrowd){
    setCoffeeInfo(coffeeCrowd);
}

```

```

public String getCoffeeShop(){
    return coffeeShop;
}

```

```

public String getCoffeeShopURL(){
    return coffeeShopURL;
}
}

```

FindCoffeeActivity.java

In FindCoffeeActivity.java we first need to create an object of our new CoffeeShop class and then implement a findCoffee() method.

```

private CoffeeShop myCoffeeShop = new CoffeeShop();

```

```

private void findCoffee(View view){
    //get spinner
    Spinner crowdSpinner = findViewById(R.id.spinner);
    //get spinner item array position
    Integer crowd = crowdSpinner.getSelectedItemPosition();
    //set the coffee shop
    myCoffeeShop.setCoffeeShop(crowd);
    //get suggested coffee shop
}

```

```
String suggestedCoffeeShop = myCoffeeShop.getCoffeeShop();
//get URL of suggested coffee shop
String suggestedCoffeeShopURL = myCoffeeShop.getCoffeeShopURL();
Log.i("shop", suggestedCoffeeShop);
Log.i("url", suggestedCoffeeShopURL);
}
```

For now we're just logging the results for testing.

Button

We could add the onClick event to our button like we've been doing, but we're going to use an event listener instead.

You can only use the android:onClick attribute in activity layouts for buttons, or any views that are subclasses of Button such as CheckBoxes and RadioButtons. So it's good to understand how to set up event listeners.

<https://developer.android.com/guide/topics/ui/ui-events>

An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

The onClick() callback method is called from the View.OnClickListener event listener. This event is fired when the user either touches the item.

To define the method and handle the event, implement the nested interface in your Activity. Then, pass an instance of your implementation to View.setOnClickListener() method.

Update the onCreate() method.

@Override

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_find_coffee);
    //get button
    Button button = findViewById(R.id.button);
    //create listener
    View.OnClickListener onclick = new View.OnClickListener(){
        public void onClick(View view){
            findCoffee(view);
        }
    };
    //add listener to the button
    button.setOnClickListener(onclick);
}
```

Now run your project and look in the Logcat to see if it's working.

New Activity

File | New | Activity

Either Gallery or Empty Activity

Activity name: ReceiveCoffeeActivity

Check Generate Layout File

Layout Name: activity_receive_coffee

Do not check Launcher Activity as this is not the launcher activity for our app.

This creates a new layout xml file and java file for our new activity.

It also updates the AndroidManifest.xml file with a new activity.

Our layout will simply consist of a textView where we'll suggest a coffee shop.

Add a text view and give its text a string resource and a descriptive id of coffeeShopTextView (you can remove the text once your layout is set)

```
<string name="suggested_coffee">This is your suggested coffee shop</string>
```

Explicit Intent

Now let's get the button in the first activity to call the second activity.

In FindCoffeeActivity.java update findCoffee() to create and start an intent.

```
//create an intent
```

```
Intent intent = new Intent(this, ReceiveCoffeeActivity.class);
```

[note: AS is going to add "packageContext:" before this)

Before we start the intent, let's pass data to it.

(press option + return on Mac (Alt + Enter on Windows) to import missing classes.)

Passing Data

Now let's pass the coffee shop name and URL to the second activity.

In FindCoffeeActivity.java BEFORE you start the new activity, add the data to the intent.

```
//pass data
```

```
intent.putExtra("coffeeShopName", suggestedCoffeeShop);
```

```
intent.putExtra("coffeeShopURL", suggestedCoffeeShopURL);
```

[note: AS is going to add "name:" before your String key)

```
//start the intent
```

```
startActivity(intent);
```

Receiving Data

Now let's update ReceiveCoffeeActivity.java to get the data sent in the intent. Create two private strings in the class

```
private String coffeeShop;
```

```
private String coffeeShopURL;
```

The onCreate() method is called as soon as the activity is created so that's where we'll get the intent.

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_receive_coffee);
```

```
    //get intent
```

```

Intent intent = getIntent();
coffeeShop = intent.getStringExtra("coffeeShopName");
coffeeShopURL = intent.getStringExtra("coffeeShopURL");
Log.i("shop received", coffeeShop);
Log.i("url received", coffeeShopURL);

//update text view
TextView messageView = findViewById(R.id.coffeeShopTextView);
messageView.setText("You should check out " + coffeeShop);
}

```

Make sure that the string you're using in `getStringExtra()` is EXACTLY the same as the string you used in `putExtra()` in `FindCoffeeActivity.java`.

Implicit Intent

Let's add a button in our second activity to open up the coffee shop's web site in an external app.

Add an image button in the bottom right corner of the layout.

Add an image resource into the drawable folder and use that for the button's src.

By default image buttons have a background. You can change its color or make it transparent.

`android:background="@android:color/transparent"`

In `ReceiveCoffeeActivity.java` implement a method to load a web page.

```

private void loadWebSite(View view){
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse(coffeeShopURL));
    startActivity(intent);
}

```

`Uri.parse()` parses the string passed to it and creates a `Uri` object. A `Uri` object is an immutable reference to a resource or data.

Update `onCreate()` to set up the event listener and add it to the image button.

```

ImageButton imageButton = findViewById(R.id.imageButton);
//create listener
View.OnClickListener onclick = new View.OnClickListener() {
    public void onClick(View view) {
        loadWebSite(view);
    }
};
//add listener to the button
imageButton.setOnClickListener(onclick);

```

Run and use the back arrow to see how it takes you to the last activity you were in.

ATLS 4120: Mobile Application Development

Week 12: Android Lifecycle

In Android an activity is a single, specific task a user can do. Most apps include several different activities that allow the user to perform different actions. Generally, one activity implements one screen in an app.

When the user taps the app icon on the Home screen, it starts the main activity in the Android manifest file. By default this is the activity you defined when you created your Android project. Every app must have an activity that is declared as the launcher activity. This is the main entry point to the app

- The main activity for your app must be declared in the `AndroidManifest.xml` file
- Must have an `<intent-filter>` that includes the MAIN action and LAUNCHER category
- When you create a new Android project the default project files include an Activity class that's declared in the manifest with this filter.
- If either the MAIN action or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

[show this in the Halloween app]

Activities transition through different states throughout their life cycle that are important to understand.

Android Activity States (slide)

- Created
 - An app's main activity has been launched
 - Your activity does not reside in the Created state, it then enters the *Started* state.
- Started
 - Activity is becoming visible
 - As with the Created state, the activity does not stay resident in the Started state, it then enters the *Resumed* state.
- Resumed
 - App is visible in the foreground and the user can interact with it, the running state
 - This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app.
- Paused
 - The activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode).
 - User is leaving the activity (though it does not always mean the activity is being destroyed)
 - Activity is partially visible, another activity is in the foreground
 - When paused it does not receive user input and doesn't execute any code
- Stopped
 - Activity is in the background and no longer visible
- Destroyed
 - All app processes have ended

Android has callback methods that correspond to specific stages of an activities lifecycle. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

Android Lifecycle Methods

<https://developer.android.com/guide/components/activities/activity-lifecycle.html>

The lifecycle methods are all from the Activity class

- **onCreate()** – activity is first created
 - Good place to initialize the essential components of your activity and do setup
 - calls **setContentView()** to define the layout for the activity's user interface
 - **setContentView(R.layout.activity_main)** is how the activity knows which layout to load in the view
 - Your activity does not reside in the Created state. After the [onCreate\(\)](#) method finishes execution, the activity enters the *Started* state, and the system calls the [onStart\(\)](#) and [onResume\(\)](#) methods in quick succession.
- **onStart()** – activity is becoming visible
 - **onStart()** contains the activity's final preparations for coming to the foreground and becoming interactive.
 - Followed by **onResume()** if the activity comes into the foreground
 - Followed by **onStop()** if the activity is made invisible
 - The [onStart\(\)](#) method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the [onResume\(\)](#) method.
- **onResume()** – activity is in the foreground
 - the system calls **onResume()** every time your activity comes into the foreground, including when it's created for the first time and when the app goes from Paused to Resumed
 - This is where the lifecycle components can enable any functionality that needs to run while the component is visible and in the foreground.
 - When an interruptive event occurs, the activity enters the *Paused* state, and the system invokes the [onPause\(\)](#) callback.
- **onPause()** – activity is no longer in the foreground, another activity is starting
 - The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode).
 - An activity in the Paused state may continue to update the UI if the user is expecting the UI to update (ie Google Maps)
 - Release or adjust any operations or release any resources not needed when paused
 - [onPause\(\)](#) execution is very brief, and does not necessarily afford enough time to perform save operations. Instead, you should perform heavy-load shutdown operations during [onStop\(\)](#).
 - Followed by **onResume()** if the activity returns to the foreground
 - Followed by **onStop()** if the activity becomes invisible
- **onStop()** – activity is no longer visible

- Use **onStop()** to perform large, CPU intensive operations such as saving application or user data, make network calls, or execute database transactions.
- Followed by **onRestart()** if the activity becomes visible again
- Followed by **onDestroy()** if the activity is going to be destroyed
- If the device is low on memory **onStop()** might not be called before the activity is destroyed
- **onRestart()** – activity was stopped and is about to restart
 - **onRestart()** doesn't get called the first time the activity is becoming visible so it's more common to use the **onStart()** method
- **onDestroy()** – activity is about to be destroyed
 - The system invokes this callback either because:
 - the activity is finishing (due to the user completely dismissing the activity or due to finish() being called on the activity)
 - the system is temporarily destroying the activity due to a configuration change (such as device rotation or multi-window mode)
 - The **onDestroy()** callback should release all resources that have not yet been released by earlier callbacks such as **onStop()**.

When you override these methods to implement them make sure you call their super class method first.

Pausing and Resuming

- When an activity in the foreground becomes partially obscured it becomes paused
 - Stop ongoing actions
 - Commit unsaved changes (if expected)
 - Release system resources
- As long as an activity is partially visible but not in focus it remains paused
- When the user resumes the activity you should reinitialize anything you released when it paused

When your activity is paused, the Activity instance is kept in memory and is recalled when the activity resumes.

Stopping and Restarting

- If an activity is fully obstructed and not visible it becomes stopped
 - Switches to another app
 - Another activity is started
 - User gets a phone call
- The activity remains in memory while stopped
- If the user goes back to the app, or uses the back button to go back to the activity, it is restarted

When your activity is stopped, the Activity instance is kept in memory and is recalled when the activity resumes.

You don't need to re-initialize components that were created during any of the callback methods leading up to the Resumed state.

The system also keeps track of the current state for each View in the layout.

Destroying and Recreating

- An activity is destroyed when the system decides it's no longer needed
 - User presses the back button
 - Hasn't been used in a long time
 - Needs to recover memory
- A change in device orientation results in the activity being destroyed and then recreated so the new device configuration can be loaded
 - Device configuration includes screen size, screen orientation, whether there's a keyboard attached, and also configuration options specified by the user (such as the locale).
- If the system destroys the activity due to system constraints, then although the actual Activity instance is gone, the system saves some state data in a Bundle object
 - If the user navigates back to that activity, a new instance of the activity is recreated using the data saved in the Bundle object
- As an activity begins to stop, the system calls the **onSaveInstanceState(Bundle)** method to save the current state of the activity
 - By default the Bundle instance saves information about each View object in your activity layout (such as the text in an EditText)
 - To save additional data use the **savedInstanceState.putxxx()** methods
 - putInt()
 - putBoolean()
 - putLong()
 - etc
 - A Bundle object isn't appropriate for preserving more than a trivial amount of data because it requires serialization on the main thread and consumes memory.
- The **onCreate()** method is called whether the system is creating a new instance of your activity or recreating a previous one, so it's a good place to restore your saved state data by accessing the Bundle using the **savedInstanceState.getxxx()** methods
 - Instead of restoring the state during **onCreate()** you may choose to implement **onRestoreInstanceState()** which the system calls after the **onStart()** method.

Halloween

Let's update Halloween so the state is saved when the device is rotated and the activity is destroyed. (Halloween theme constraints state)

In MainActivity.java we had everything in our sayBoo() method. But now we'll need access to the message, the TextView, and the ImageView in multiple methods, so we'll make them instance variables.

private String **message**;

private TextView **booText**;

private ImageView **ghost**;

It's important to understand that at this level we can't access anything in the layout file, such as findViewById(), because our view hasn't been loaded yet. That happens in onCreate() when setContentView() is called.

Update booText() so the entire message is saved in our message variable.

```
message = "Happy Halloween " + nameValue + "!";  
booText.setText(message);
```

First we'll save our state, our message and image, when the activity is destroyed. You can just start typing onSave... and you'll be able to pick the method header for onSaveInstanceState(Bundle outState)

// invoked when the activity may be temporarily destroyed, save the instance state here

```
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    outState.putString("msg", message);  
    outState.putInt("imageid", R.drawable.ghost);  
  
    super.onSaveInstanceState(outState);  
}
```

Now we have to update onCreate() so we check to see if there's a savedInstanceState. Notice that onCreate() is passed a Bundle object containing the activity's previously saved state. If the activity has never existed before, the value of the Bundle object is null.

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    booText = findViewById(R.id.message);  
    ghost = findViewById(R.id.imageView);  
  
    // check for recovering the instance state  
    if (savedInstanceState != null) {  
        message = savedInstanceState.getString("msg");  
        int image_id = savedInstanceState.getInt("imageid");  
  
        booText.setText(message);  
        ghost.setImageResource(image_id);  
    }  
}
```

We must set the content view to our layout before we can access anything in our layout. We call findViewById() to get access to our TextView and ImageView so we can set their values.

If savedInstanceState exists, we recover the state using the same keys we used when we saved it. Then we update our layout with these values.

ATLS 4120: Mobile Application Development

Week 14: Device Environments

Notes for project 2:

- Floating action button (FAB) for insert
- Long press for delete
- Along with handling the UI for landscape orientation, consider any state you need to save as well since the activity is destroyed and then recreated when a device is rotated.

With Android running on so many different size devices your app layouts must be flexible and adapt to different size screens and orientations.

Supporting different screens

<https://developer.android.com/training/multiscreen/screensizes>

Android categorizes device screens using two general properties: size and density. The screen orientation (landscape or portrait) is considered a variation of screen size. Android automatically scales your layout in order to properly fit the screen. So your layouts for different screen sizes don't need to worry about the absolute size of UI elements but instead focus on the layout structure that affects the user experience (such as the size or position of important views relative to sibling views).

Constraint Layout

Constraint layout creates a layout responsive for different screen sizes.

Avoid hard coded layout values

- "wrap_content" tells the view to set its size to whatever is necessary to fit the content within that view.
- "match_parent" makes the view expand to as much as possible within the parent view.

Create alternative layouts

If you can't define one layout that works for all screen sizes or orientations you'll need to define different layouts for each configuration orientation.

Remember that the default layout is in the res/layout folder.

Screen size qualifiers:

You can provide screen-specific layouts by creating additional res/layout/ directories and then append a screen configuration qualifier to the layout directory name for each screen configuration that requires a different layout

- layout-w600dp for screens that have 600dp of available width

You can also use the smallest width qualifier (sw) to provide alternative layouts for screens that have a minimum width measured in density-independent pixels

- layout-sw600dp for screens that have a minimum available width of 600dp

Orientation qualifier:

To provide different layouts based on the device's orientation you can add the "port" or "land" qualifiers to your resource directory names. Just be sure these come *after* the other size qualifiers.

- layout-land for devices in landscape orientation
- layout-sw600dp-land for devices with 600dp wide and larger and in landscape orientation

Supporting different languages

We've been using IDs in our layout xml files and assigning those strings in our res/values/strings.xml file, which is the default strings file. If this default file is absent, or if it is missing a string that your application needs, then your application will not run and will show an error.

To add support for more languages, create additional values directories inside res/ that include a hyphen and the ISO language code at the end of the directory name. Android loads the appropriate resources according to the locale settings of the device at run time. You can do this for drawables too.

- values-es/ is the directory containing simple resources for the locale with the language code "es".

When a user runs an app Android selects which resources to load based on the device's locale. If it finds a resource for that locale it will use the resources in it. If it doesn't find a resource in that file it will look in the default resource file. If it can't find the resource in any file the app won't load.

Coffee

In the Finder make a copy of your coffee project and change the name of the folder (Coffee Adapt)

Go into our coffee app and see how the first activity looks in landscape mode.

Using constraint layout it seems that all the constraints were to the margins so in landscape the button and image were up too high. I changed a few constraints by deleting the constraints to the margins and replaced them so that they were relative to other widgets using **app:layout_constraintTop_toBottomOf** and that fixed them a bit.

Landscape variation

Now we're going to see how you define a new layout for landscape mode.

<https://developer.android.com/studio/write/layout-editor.html#create-variant>

Open the layout file

Design mode or preview

Click the Orientation in Editor  in the toolbar.

Create landscape variation

In res/layout/activity_find_coffee it creates a new layout file.

In Android view you see (land) next to it.

In the Project view you can see how this is really stored in app/src/main/res/layout-land

The title of the tab has the prefix "land/"

Edit this so you have a layout that looks good in landscape.

Be careful that you know which file you're editing.

I did this by putting the button to the right of the spinner and the image view to the right of the button. First delete all constraints for the button and image view and then move them. Then use Infer Constraints for both and modify as needed.

Note: Instant Run did not seem to apply these constraint changes and I had to stop and rerun the app.

Run it and try it in both orientations. (command arrow on the Mac to rotate)

Localization

In the Project view go to app/src/main/res right click New | Android Resource Directory values-fr

(French fr, Spanish es, Japanese ja, German de. Other ISO language codes
http://www.loc.gov/standards/iso639-2/php/code_list.php)

Right click on your new folder New | Values resource file
strings.xml

Open your original strings.xml file and copy all the strings you have.

Go into your new fr/strings.xml file and paste them in the <resources> tag.

Change the string values for your new language. (There are services that do this for a fee)
(hippie had no translation so I just left it as I don't really know French.)

To test this in the emulator you need to change the language of the device.

Settings | Personal | Language & Input

Change the language to the language you're localizing to (Francais(France))

(Android also lets you define a custom locale, but we shouldn't need that)

On the Pixel: Settings | System | Languages | Add a language

Then in the list move the new language above English.

Now you'll notice that the apps that come with the phone, Settings, Camera, Phone etc are now all in the language you picked.

When you run your app you should see all your strings in the new language, but your second activity still has English.

To fix this we should make strings in our Java file IDs and define the string in the xml file just as we did with our layout. In your strings files add

<string name="message">You should check out </string>

and the French version <string name="message">Tu aimerais </string>

In ReceiveCoffeeActivity.java let's get that string resource and use it in our TextView.

```
String message = getString(R.string.message);
```

```
messageView.setText(message + coffeeShop);
```

Now run your app.

Add in a space in your message

```
messageView.setText(message + " " + coffeeShop);
```