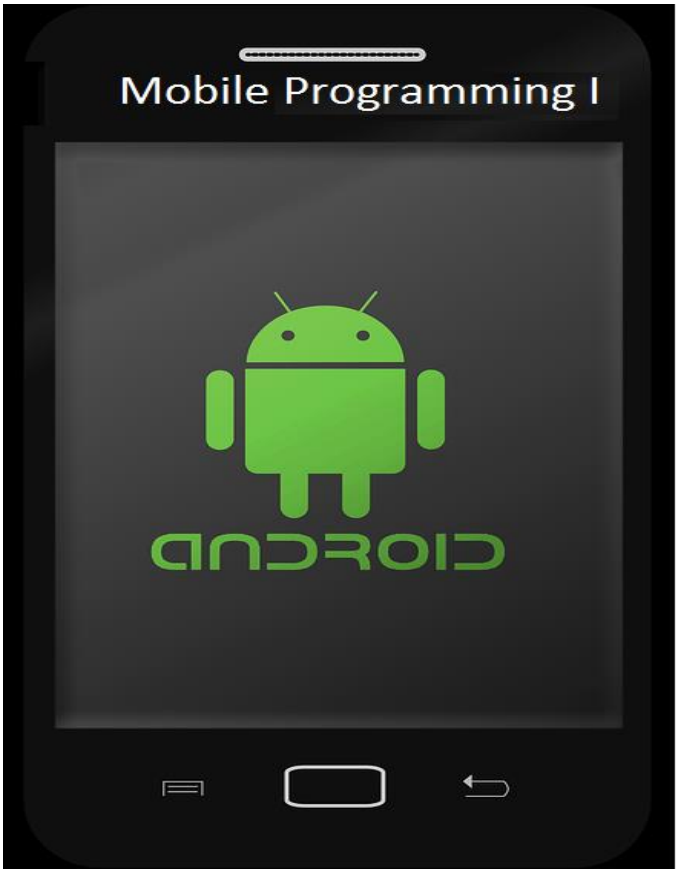
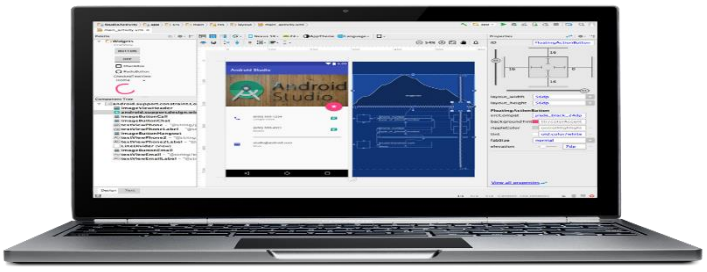


Prepared by: Asst.Lect. Mohammad Salim Abdulrahman

For

Department of IT: 4th Year Students



The contents of this course are mainly based on materials published in the Android Developers Website, Tutorial Points, and Google and Udacity courses.

(https://developer.android.com/guide/index.html)

(https://www.tutorialspoint.com/android/index.htm)

Week1

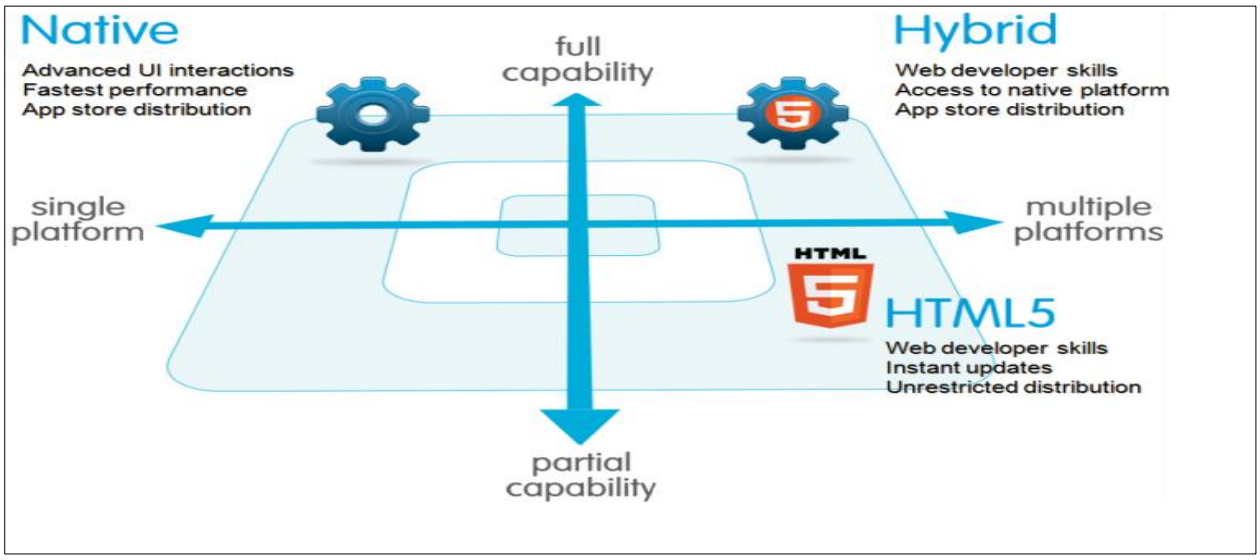
Introduction

Once you decide your mobile app idea, it's time to validate it, understand the target market, and narrow down on the platform you ideally would like to build your mobile application on. As soon as that is decided, it's time to select a programming language, keeping in mind your business strategy to make either **native, hybrid, or cross-platform apps** [1].

There are three types of apps [2]:

- **Native apps** are specific to a given mobile platform (iOS or Android) using the development tools and language that the respective platform supports Native mobile apps are developed in iOS (Objective-c or Swift), Android (Java) or Windows Phone (C#/Visual Basic). Native apps look and perform the best.
- **HTML5 apps** (**cross-platform apps**) use standard web technologies—typically HTML5, JavaScript and CSS. This write-once-run-anywhere approach to mobile development creates cross-platform mobile applications that work on multiple devices. While developers can create sophisticated apps with HTML5 and JavaScript alone, some vital limitations remain at the time of this writing, specifically session management, secure offline storage, and access to native device functionality (camera, calendar, geolocation, etc.)
- **Hybrid apps** make it possible to embed HTML5 apps inside a thin native container, combining the best (and worst) elements of native and HTML5 apps.

Mobile Apps Types [3]



Most Popular 6 Programming Languages for Mobile App Development

[4] Mobile Programming Languages

Java

Java programming language is one of the most preferred language when it comes to Android app development. An object-oriented programming language developed at Sun Microsystems (now owned by Oracle), can be run in 2 different ways either in a browser window or in a virtual machine that can do without a browser.

And this flexibility tends to mean a lot when it comes to re-using code and updating software Although Java does not have much to do if you are considering iOS development, it certainly can be on your chosen list when it comes to mobile application across platforms i.e. cross platform apps.

HTML5

HTML5 is the ideal programming language if you are looking to build a Web-fronted app for mobile devices. Although it makes various data types simple to insert, account for different screen sizes, rationalize input parameters and even level the browser playing field; the problem with HTML5 is that it is still a proposed standard. Currently supported in a lot of different ways by a lot of different browsers, HTML5 from the cost-efficiency point of view has the advantage of building on the current version of HTML — making the learning curve a much shallower than that for a completely new language.

Objective-C

The primary programming language for iOS apps, Objective-C was chosen by Apple to build apps that are robust and scalable. Being a C-language superset, it does have a number of functions that precisely deal with graphics, I/O, and display functions. Moreover, as part of the Apple development framework, Objective-C is fully integrated into all iOS and MacOS frameworks. However, it's now slowing being replaced in the Apple ecosystem by a more powerful language called Swift.

Swift

Swift is the latest programming language to foray into the Apple ecosystem, mainly considering its prevalence in writing code for Apple's latest APIs, Cocoa and Cocoa Touch. Even though it is a language written to work along with Objective-C, the Cupertino company is making it but obvious for iOS developers to turn to Swift for complete programming. Designed to eliminate the likelihood for many of the security vulnerabilities possible with Objective-C, it's time for mobile app developers to Swift, as many businesses are looking to hire Swift developer with expertise in developing cutting-edge mobile apps using the same.

C++

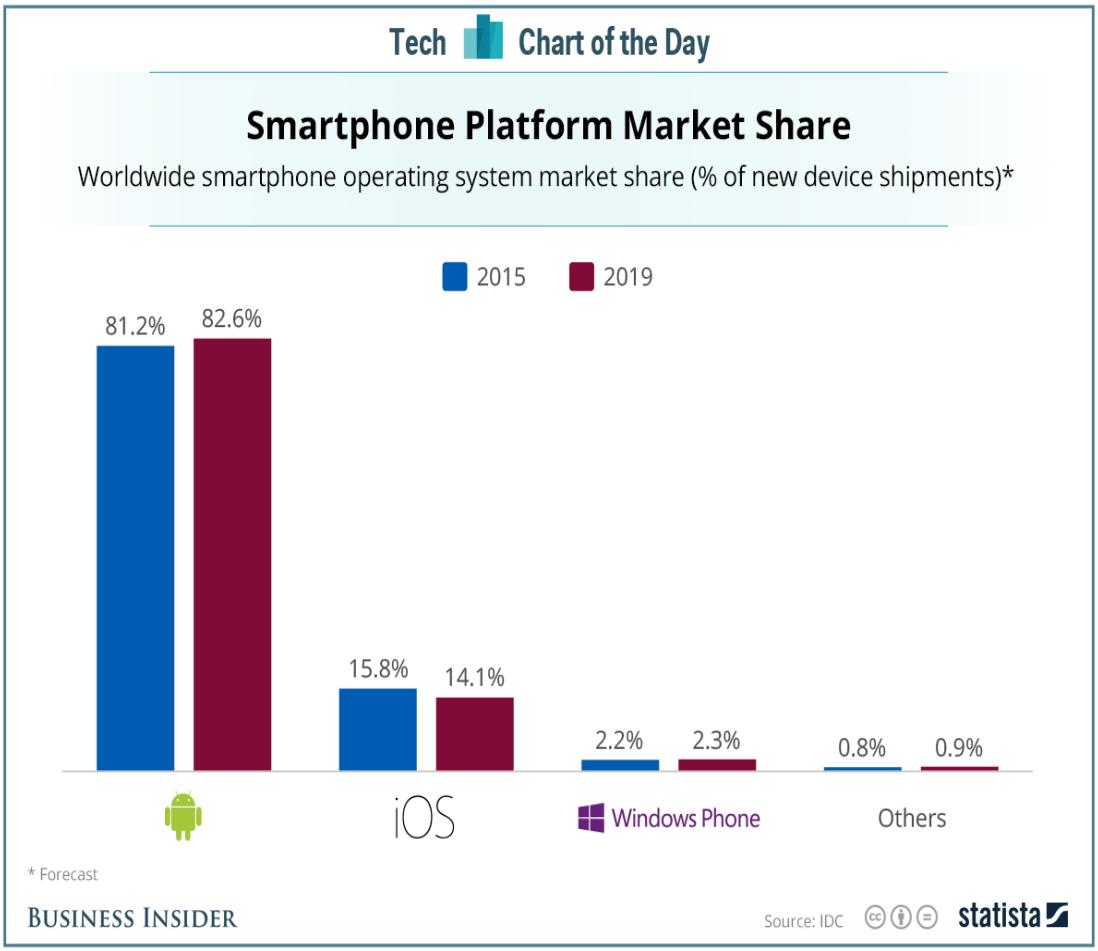
This is the most appropriate and robust programming language when it comes to building mobile apps for Android and Windows - and mainly for low-level programming it's still the go-to language on platforms for

mobile app developers. As a powerful programming language, C++ allows mobile apps to be developed for practically every purpose on every platform that exists. It might not be super stylish or trendy, but it has dominated the programming world even before the smartphone revolution.

C#

The most wanted programming language for Windows Phone app development, C# does the trick for Microsoft that Objective-C does for Apple. Although, Windows Phone platform couldn't emerge as the game changer in the mobile application development industry, for loyal Microsoft users, C# makes the perfect programming language to build the robust Windows Phone apps.

Mobile Operating Systems

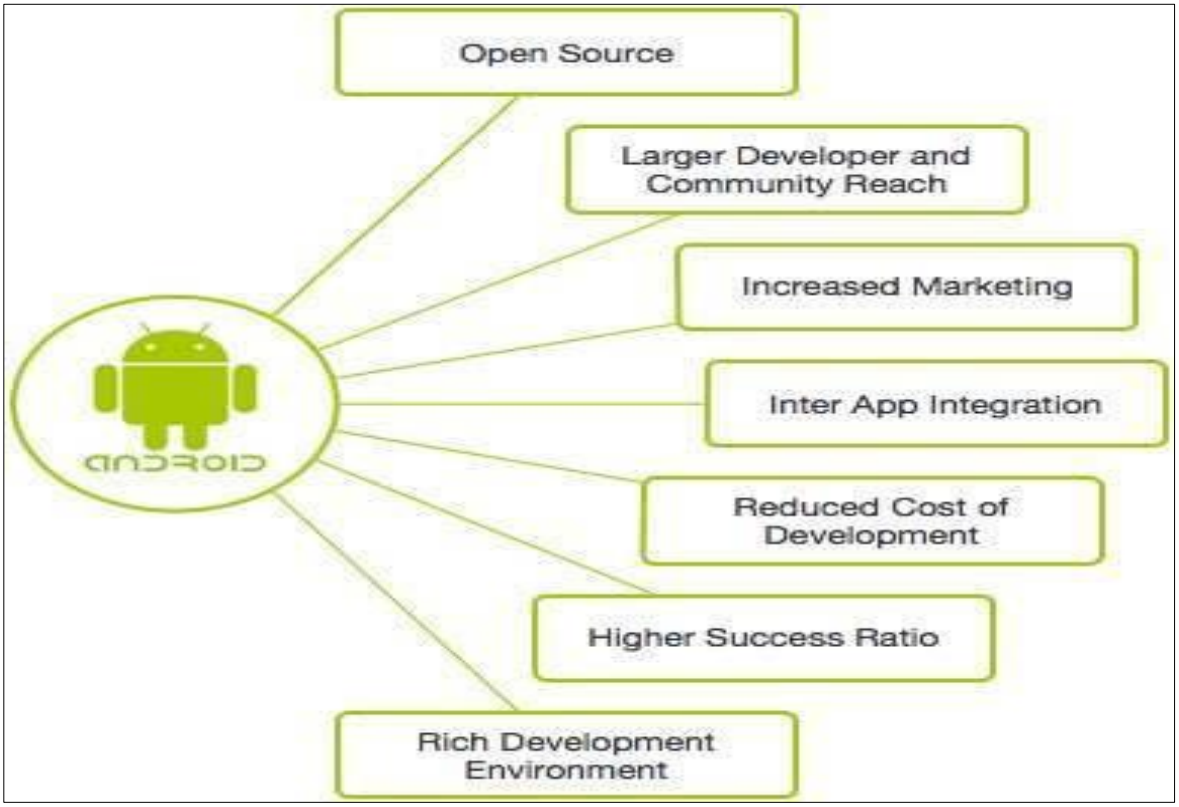


[5] Mobile OS

Android

Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.

Why Android?



Features of Android

Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below:

Feature	Description
Beautiful UI	Android OS basic screen provides a beautiful and intuitive user interface.

Connectivity	GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage	SQLite, a lightweight relational database, is used for data storage purposes.
Media support	H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
Messaging	SMS and MMS
Web browser	Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
Multi-touch	Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
Multi-tasking	User can jump from one task to another and same time various application can run simultaneously.
Resizable widgets	Widgets are resizable, so users can expand them to show more content or shrink them to save space
Multi-Language	Supports single direction and bi-directional text.
GCM	Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.

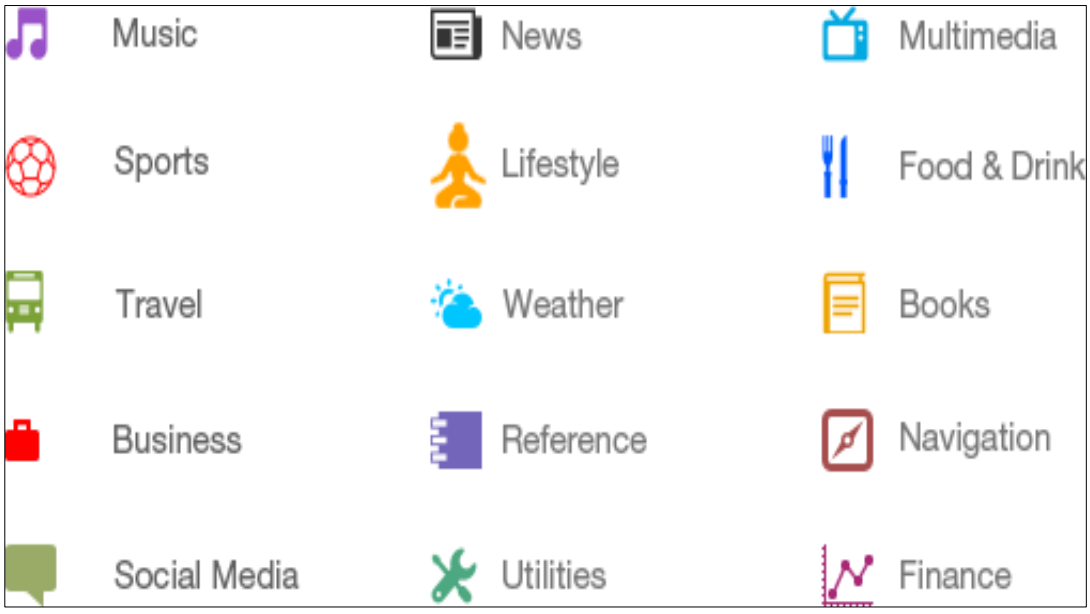
Wi-Fi Direct	A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
Android Beam	A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

Android Applications

Android applications are usually developed in the Java language using the Android Software Development Kit.

Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play**.

Categories of Android applications



History of Android

The code names of android ranges from A to O currently, let's understand the android history in a sequence.

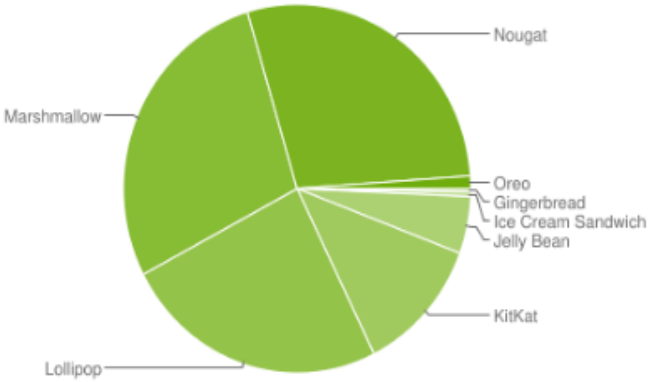
What is API level?

API Level is an integer value that **uniquely** identifies the framework API revision offered by a version of the Android platform.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 **Jelly Bean**. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

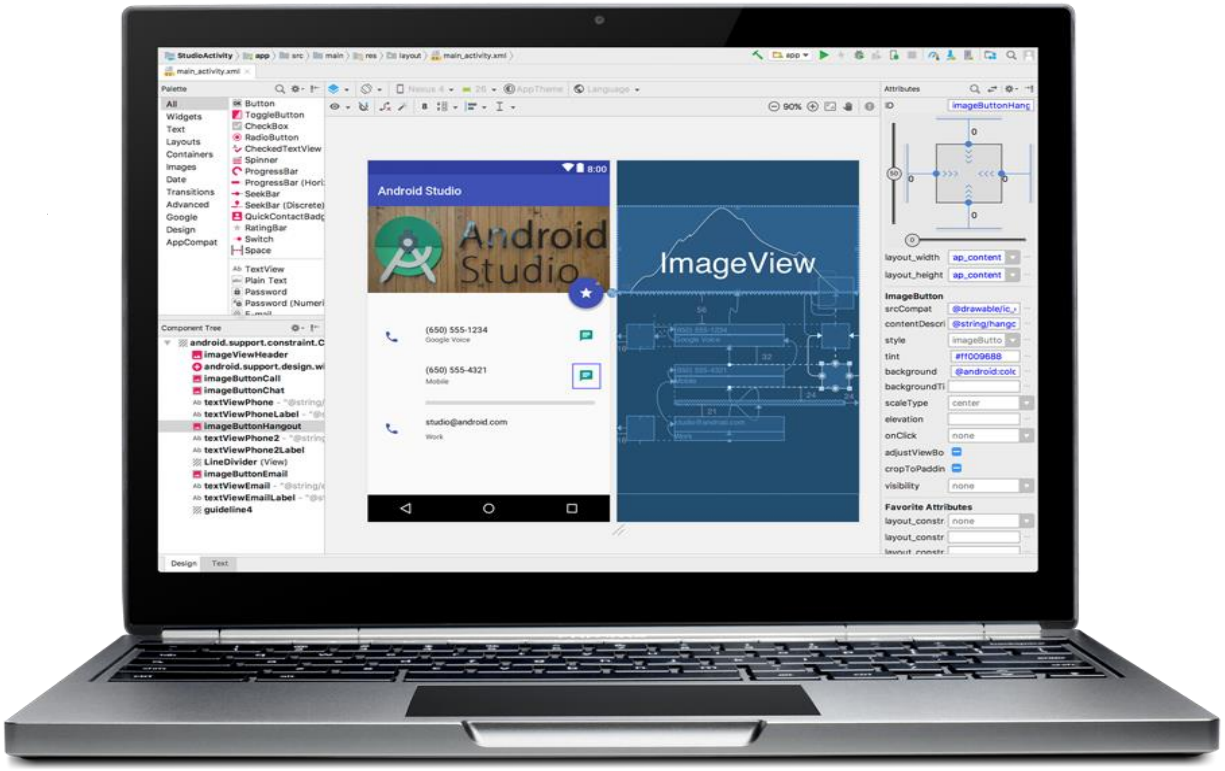
Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.4%
4.1.x	Jelly Bean	16	1.7%
4.2.x		17	2.6%
4.3		18	0.7%
4.4	KitKat	19	12.0%
5.0	Lollipop	21	5.4%
5.1		22	19.2%
6.0	Marshmallow	23	28.1%
7.0	Nougat	24	22.3%
7.1		25	6.2%
8.0	Oreo	26	0.8%
8.1		27	0.3%



Data collected during a 7-day period ending on February 5, 2018.

Android IDE

Android Studio is the Official IDE (Integrated Development Environment) for Android. It is free to download from android developer website: https://developer.android.com/studio/index.html. Its recommended to read the setup requirement before you install it to your PC.



1.Build Single Screen App

Building Layouts

Views

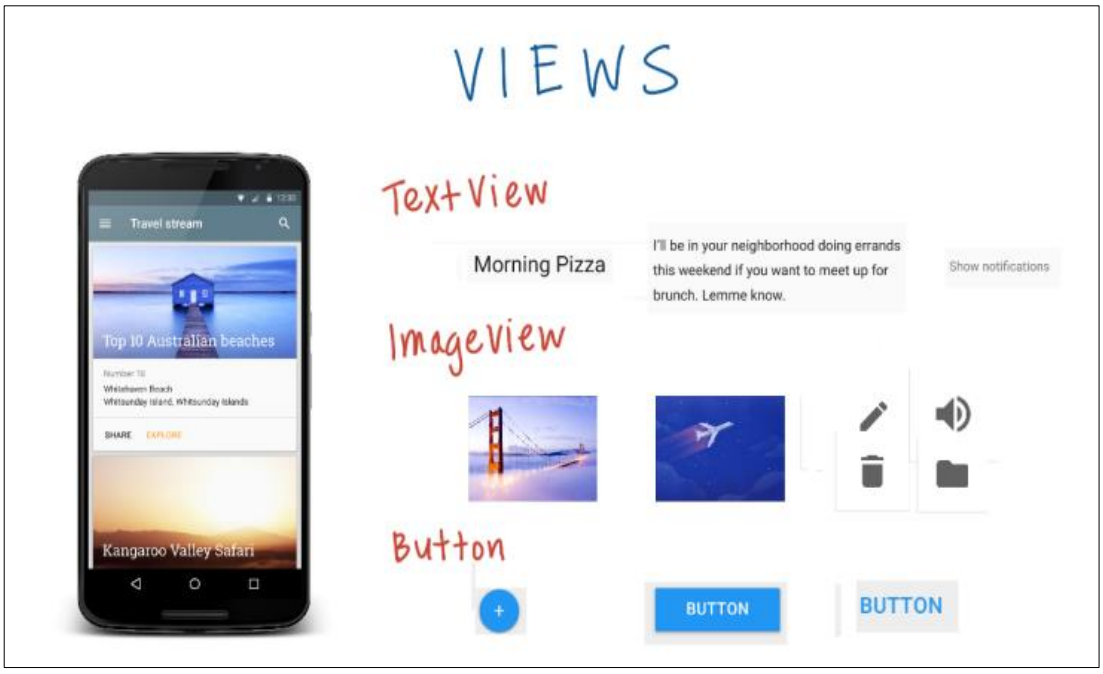
In the part, we'll move through step-by-step how to create an app, like this birthday card, and show you how to run it on an Android device.

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like **buttons, images, text fields**, etc.

Camel case is a convention that is not limited to programming. If you've ever used FedEx, listened to an iPod, created a PowerPoint, or eaten at McDonalds.

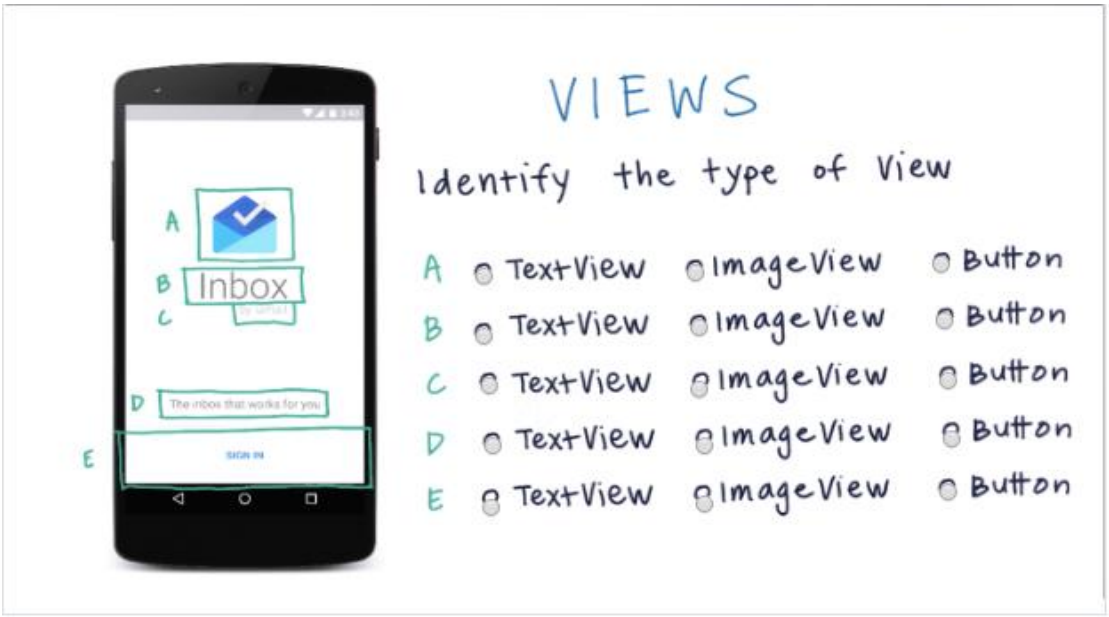
The **ViewGroup** is a subclass of **View** and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.



Example Android Views

To test you understanding do this quiz.



A layout defines the visual structure for a user interface, such as the UI for an activity or app widget **by declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

TextView

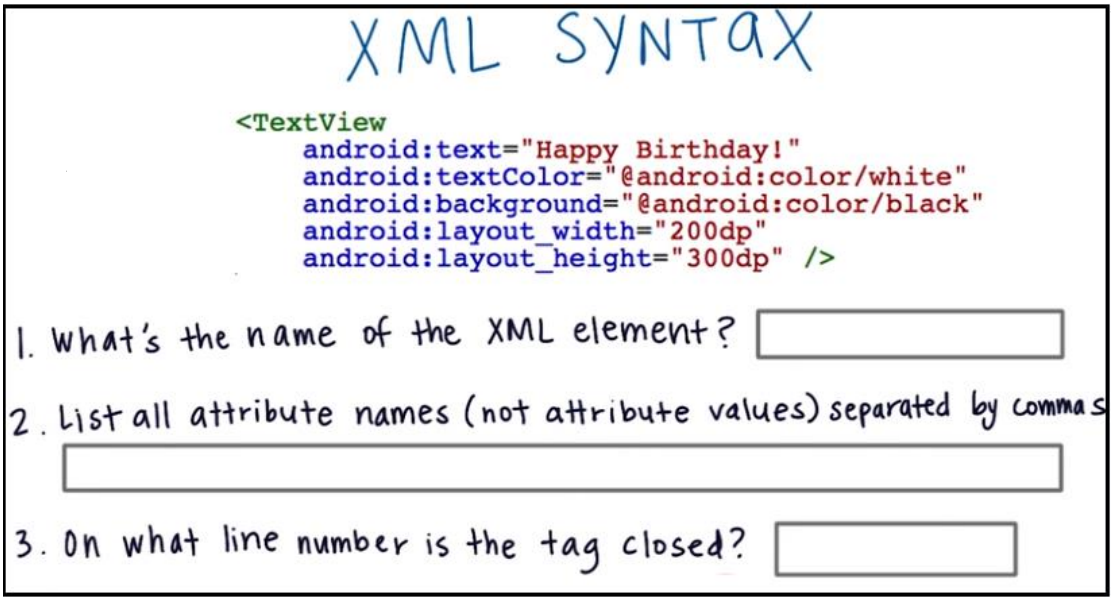
TextView is used mainly to display text on the screen, you may try the XML code below for hello world app.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello world"/>

  </RelativeLayout>
```

XML Syntax

Syntax is a set of rules that determine the valid code of XML which can be used by Android Studio to create views. Figure below shows the main XML code elements, and you may notice that colons and quotes are used.

To make it sure it's clear, try solving this quiz.

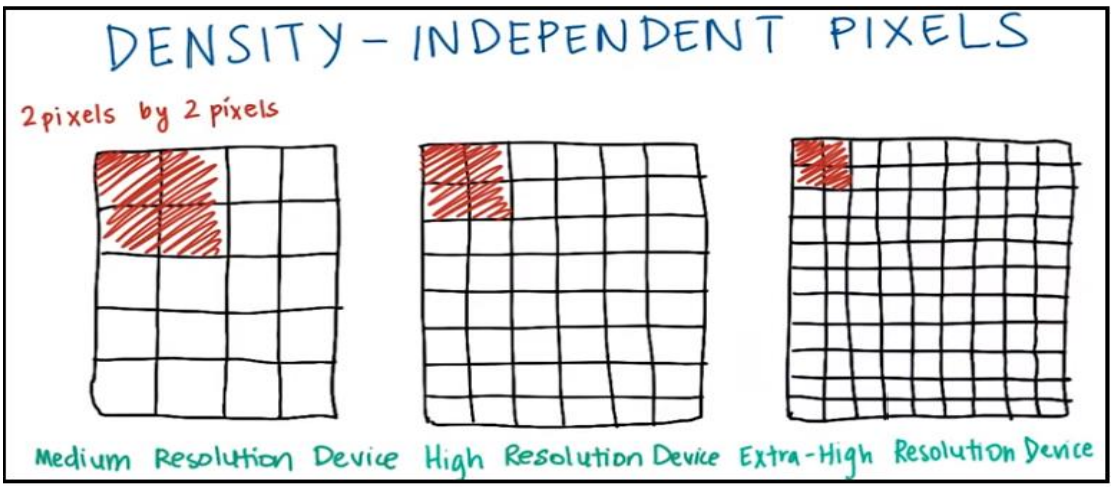


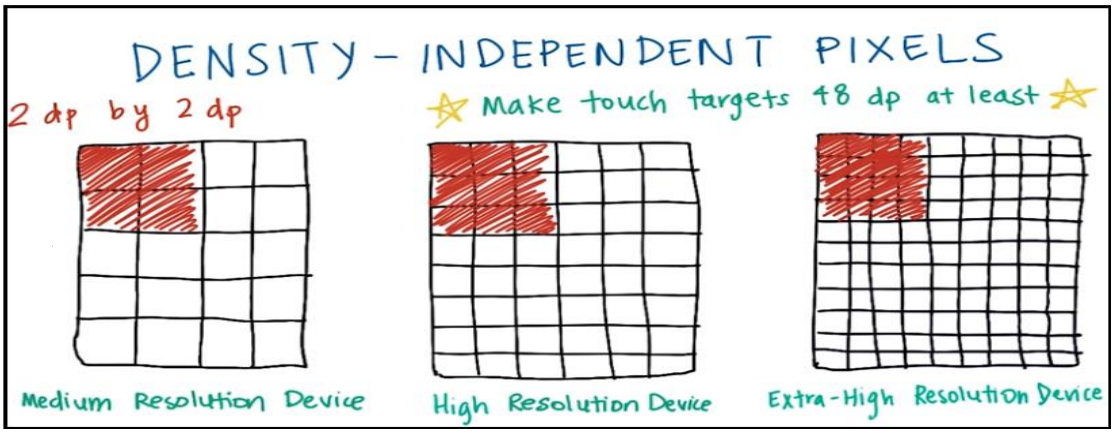
Change the TextView

What does Density Independent Pixel actually mean? You might be familiar with a pixel, which is a small illuminated area on a screen. Screens are essentially made up of hundreds of thousands of these pixels.

So what is a "density independent" pixel? Well, better screens will often have more pixels in the same amount of space. The number of pixels in a fixed space is known as the screen's pixel density. In case you're wondering, 48dp translates to approximately 9mm in physical size.

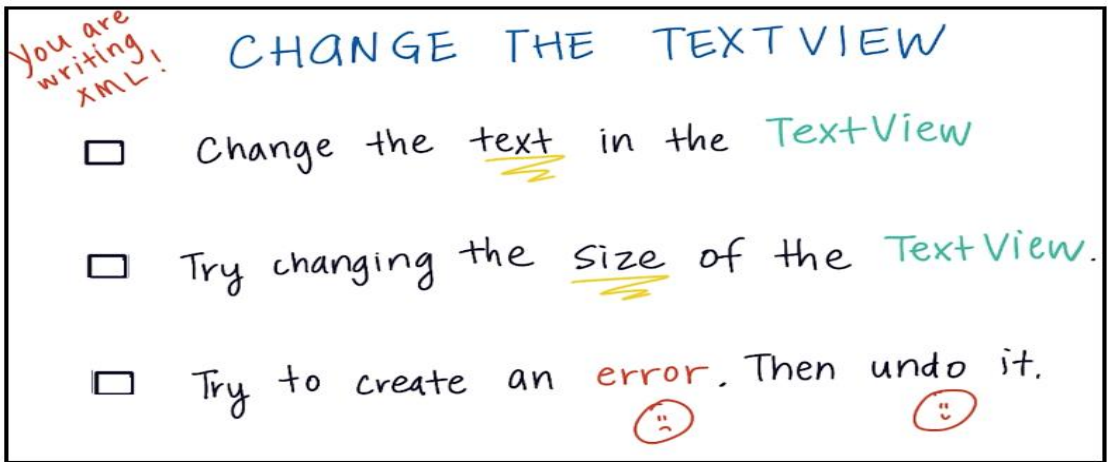
If we use just pixels we will get a problem with our design, since our TextView will be shown on different sizes across different screens.





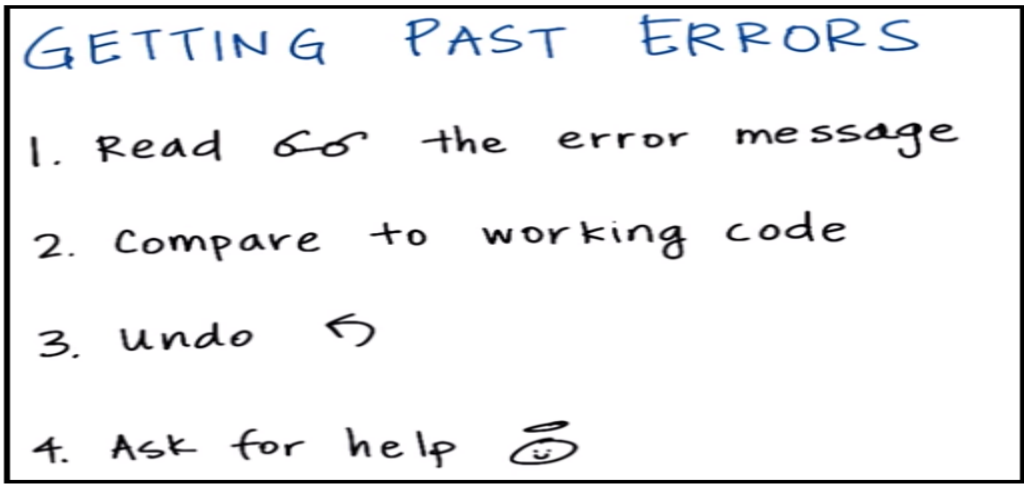
While by using **dp**, you can see in the figure above that TextView will have almost the same size across multiple devices.

To get better knowledge, do this quiz.

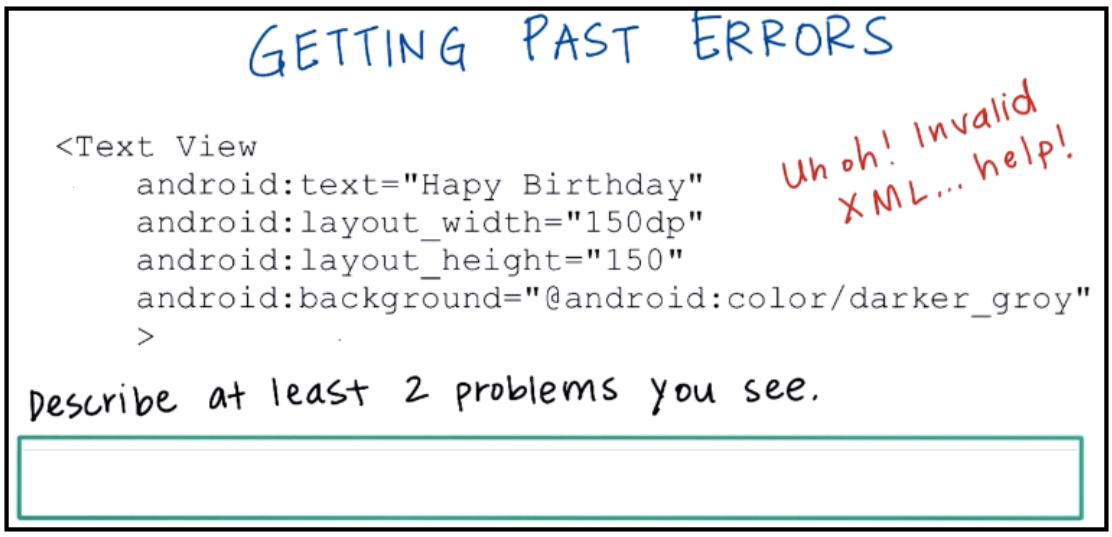


Dealing with Errors!

Here are few of most common techniques to deal with errors in your app.

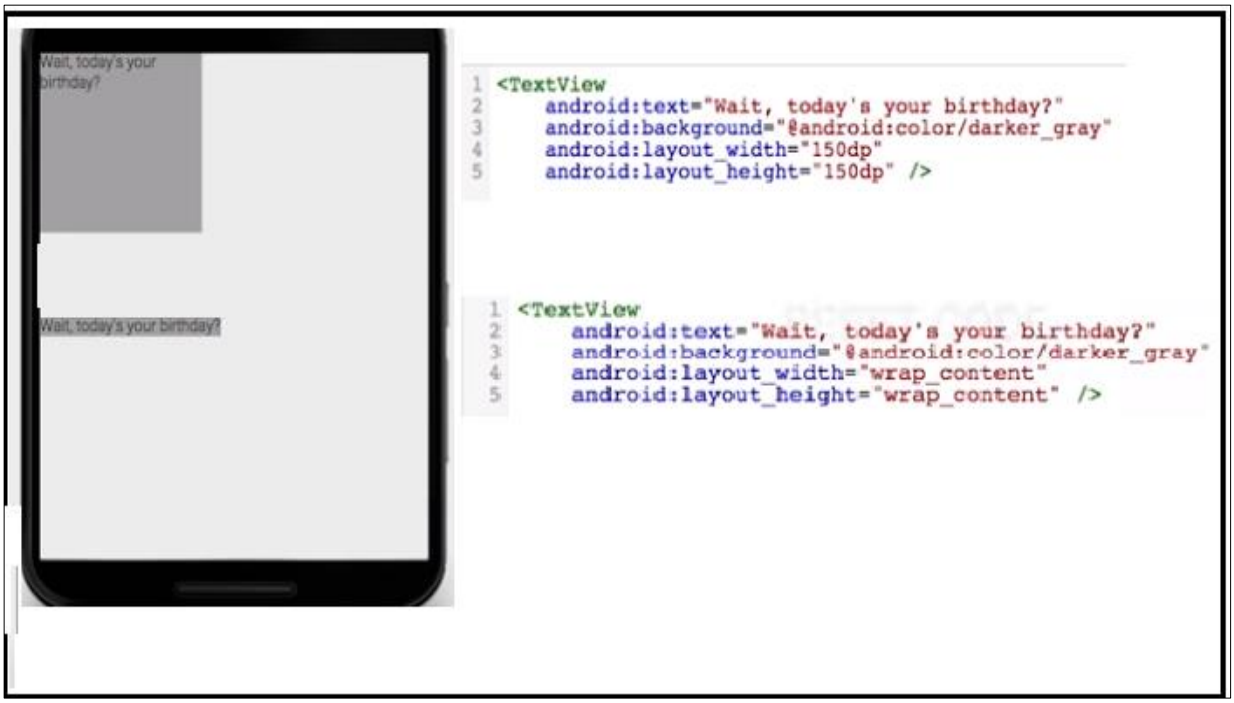


For getting a practice, solve this quiz.

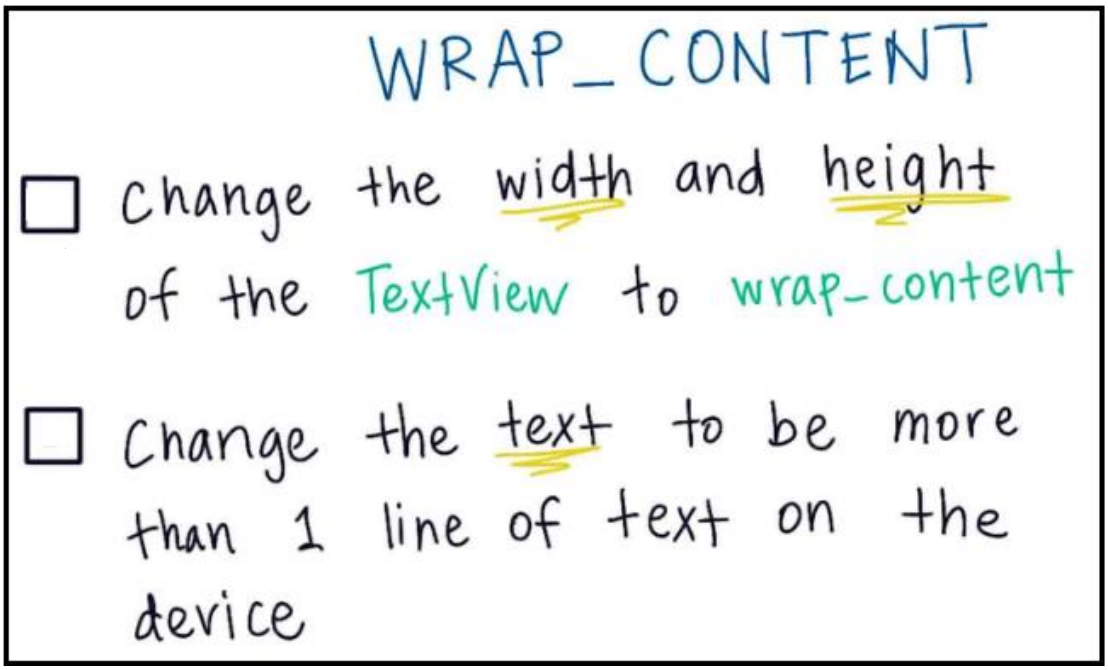


Setting Wrap Content

wrap_content value is better than using hard coding value, because the text will only the required space on the screen if wrap_content is used, other values are called hard coding which is a bad practice in designing apps due to taking more space than the needed space on screen for a particular view.



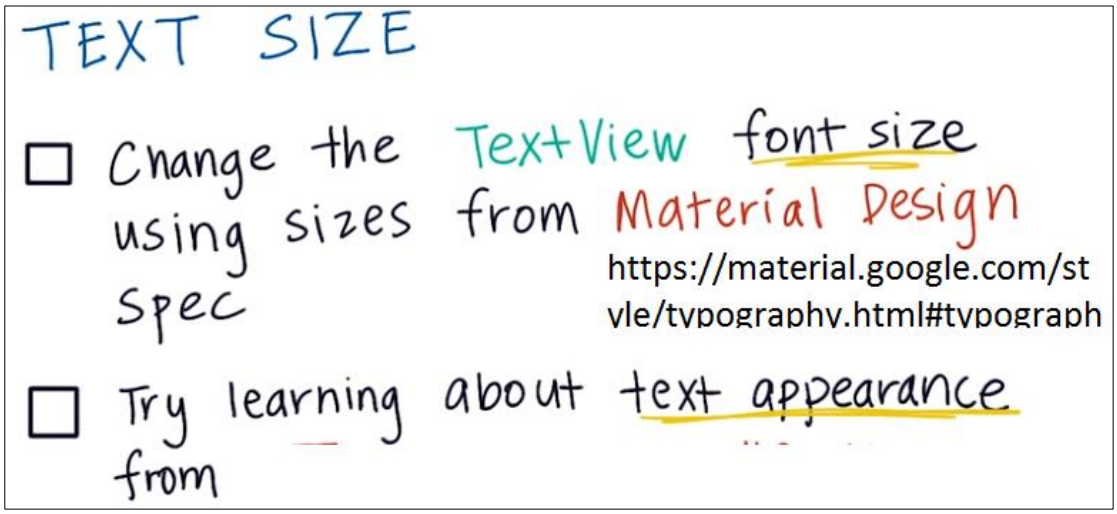
To be familiar with wrap_content , enjoy doing the quiz below.



TextView Text Size

To change textSize we use **sp**, A scale-independent pixel (sp) is a unit of length for specifying the size of a font of type. Its length depends on the user's preference for font size, set in the Settings app of the Android device. To respect the user's preferences, you should specify all font sizes in scale-independent pixels. All other measurements should be given in device-independent pixels (dp's).

If you want more information about font size, try the following quiz.



TextView Text Color

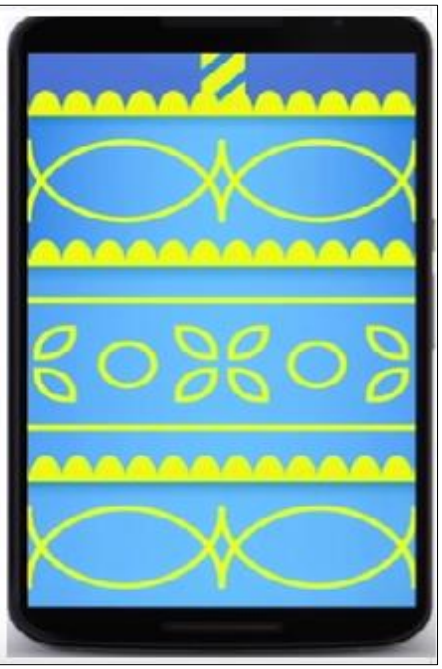
Let's go now to change text color, and you can do this as the same of this example.

To do more practice just go and change font color and background color in your code. If you need what colors you should use when designing apps , then visit **Material Design color palette** : https://material.google.com/style/color.html# or see the Hex colors at http://www.w3schools.com/colors/colors_hex.asp

Simple ImageView

When you look at the following XML code, you notice two new attributes, the fist is src which determines the name and the location of the image, while scaleType specify how you would like to display your image on the screen .

```
<ImageView
    android:src="@drawable/cake"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="center"/>
```



For example, if you use centerCrop then your image might look something like this image.

You can try learning more by doing this quiz:

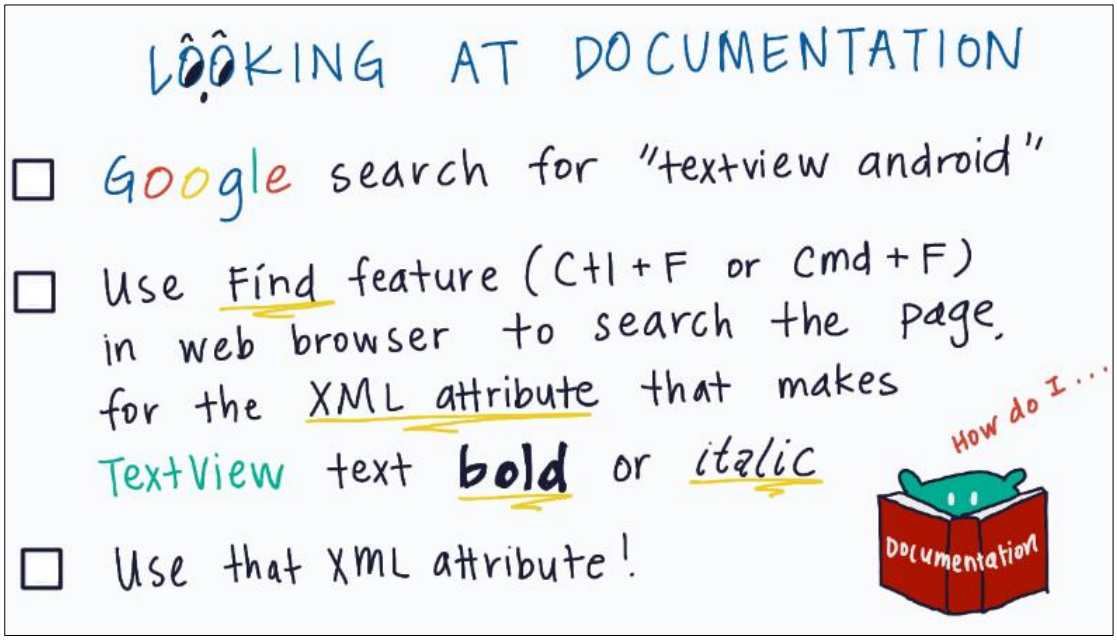
1-change width and height of the image to be fixed values such as 120dp.

2-change scaleType center and centerCrop.

3-change between different images.

Documentation

Learning anything new could be easier by using **developer.android.com**. Any website that starts with that web address is part of the official Android developer website and documentation.



2.Build Single Screen App

Building Layouts

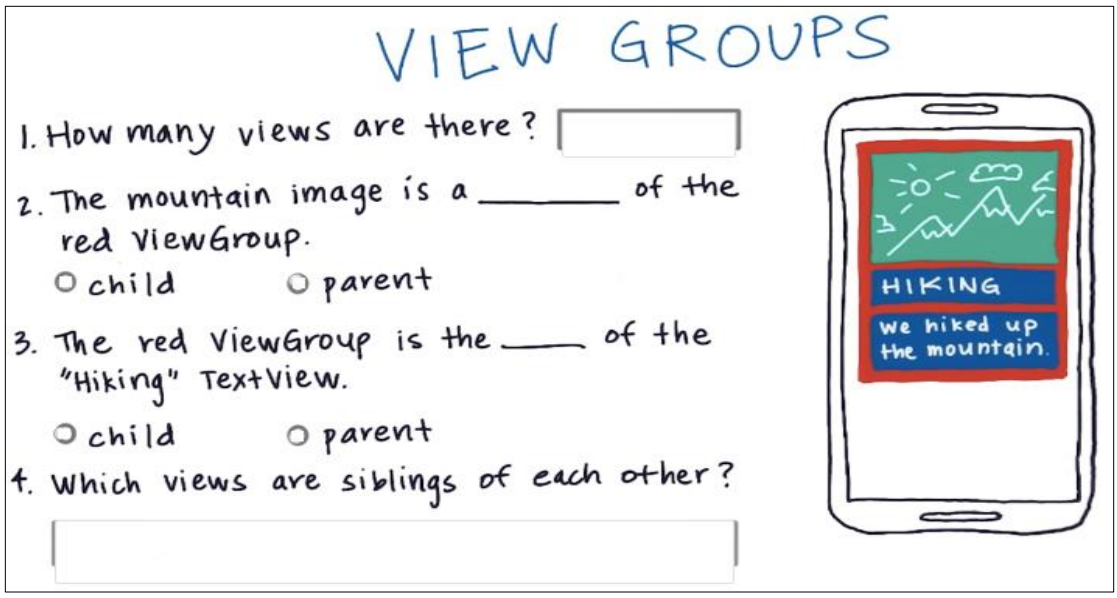
ViewGroup

A **ViewGroup** is a special view that can contain other views (called children.) The view group is the base class for layouts and views

containers. In the example the red and orange colors are the view groups.

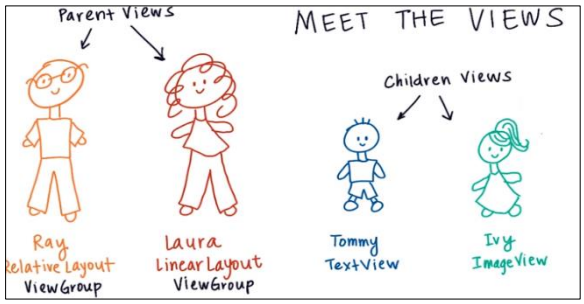
So the ViewGroup is a container of other views such as text, images and buttons. These views are considered the children of ViewGroups and they are sibling to each other.

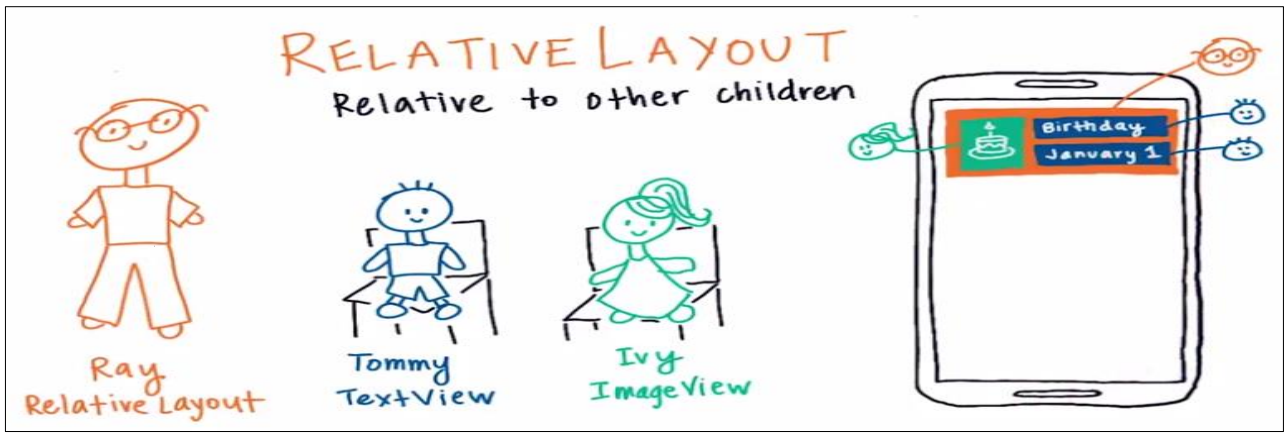
To check that you've understood View Groups, try this quiz.

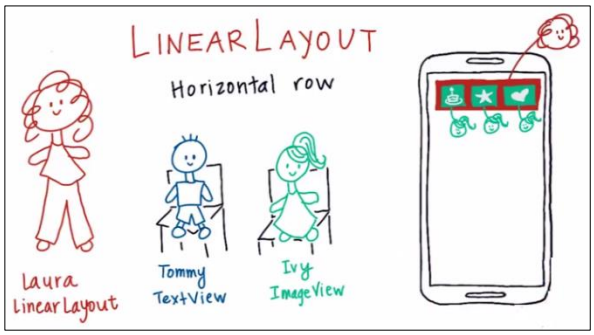


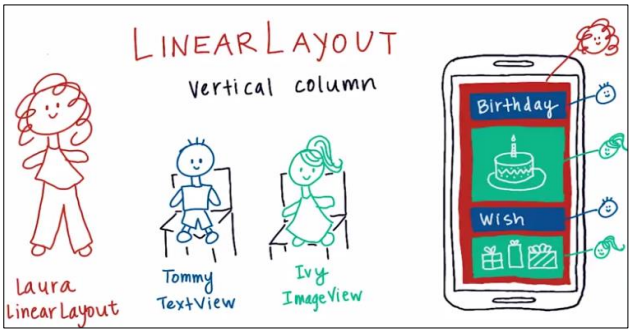
Types of ViewGroups

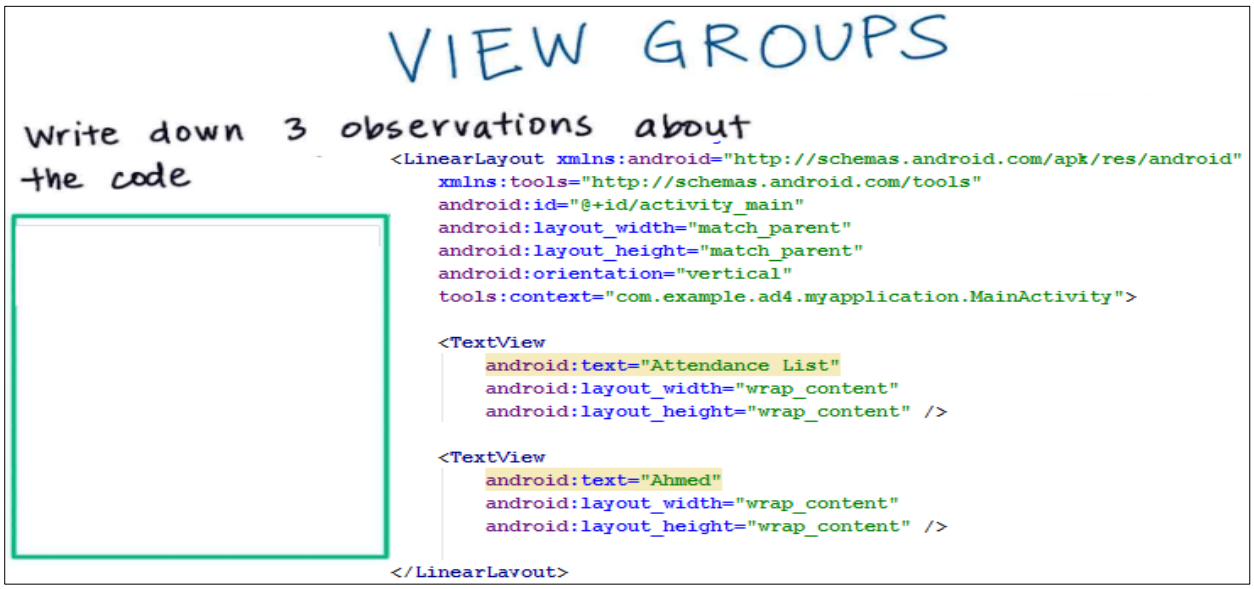
For the purpose of making types of ViewGroups more clear to you. Let's suppose that you have this family.











Linear Layout

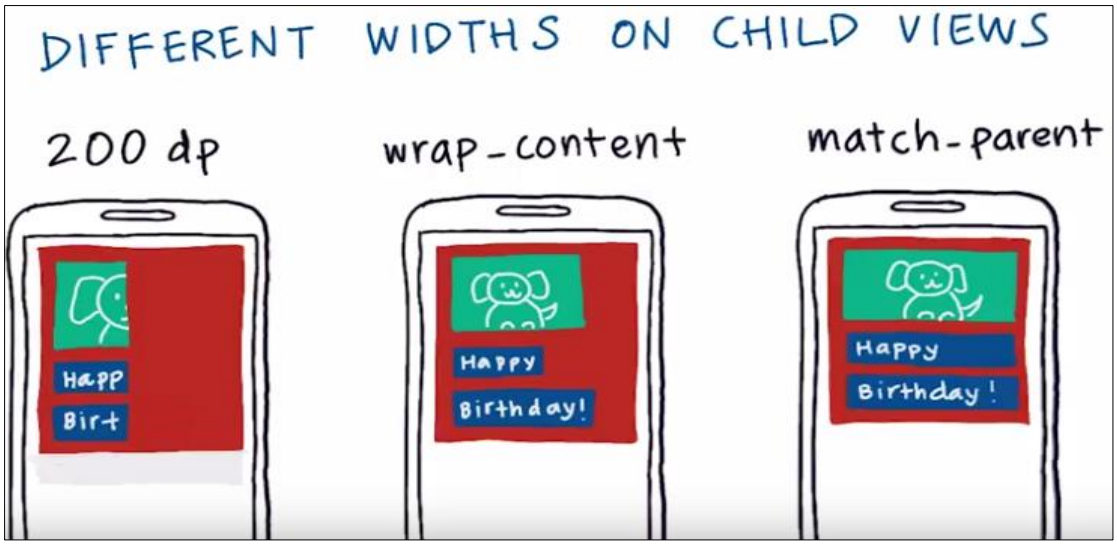
LinearLayout has two types vertical and horizontal linear layouts, and to choose any one of these types you should use the **Orientation** attribute and set its value to either **vertical** or **horizontal**.

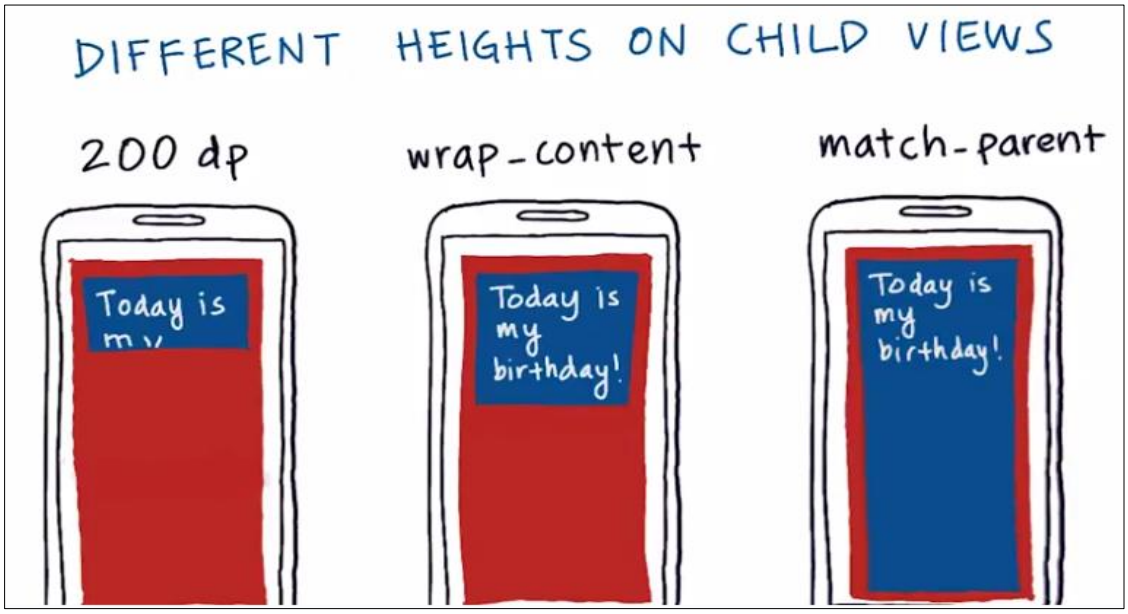
To be more familiar with LinearLayout, do practice with this short quiz:

1-Add more TextView to the parent LinearLayout view group.

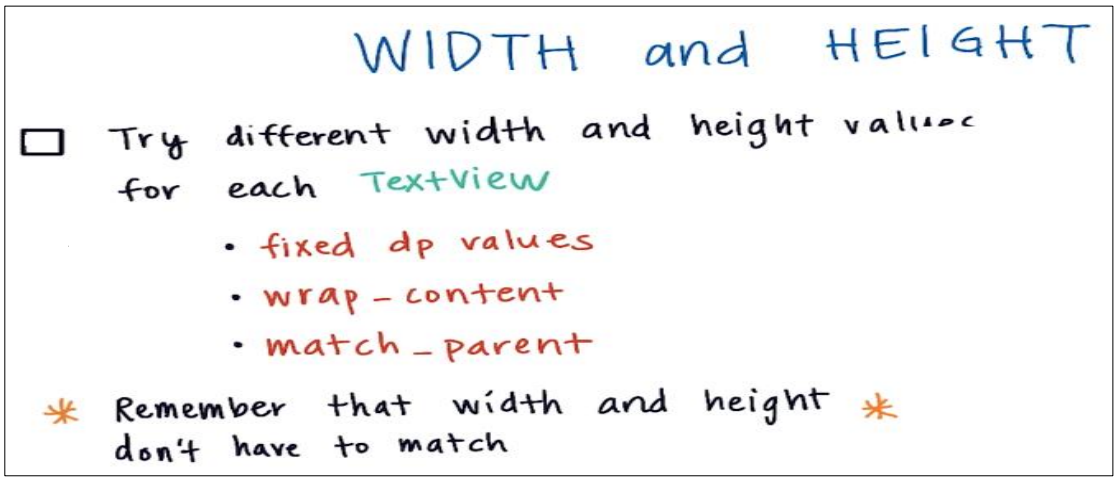
2-Change the LinearLayout orientation attribute to be vertical or horizontal.

Width or height attributes have different values as you can see in these two figures.

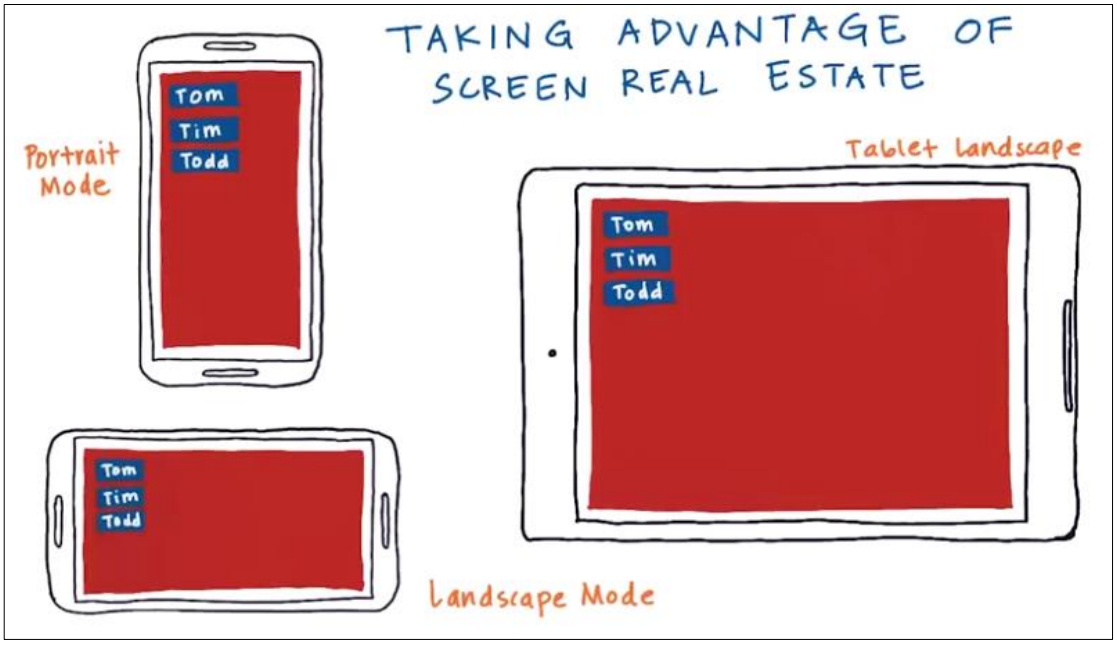




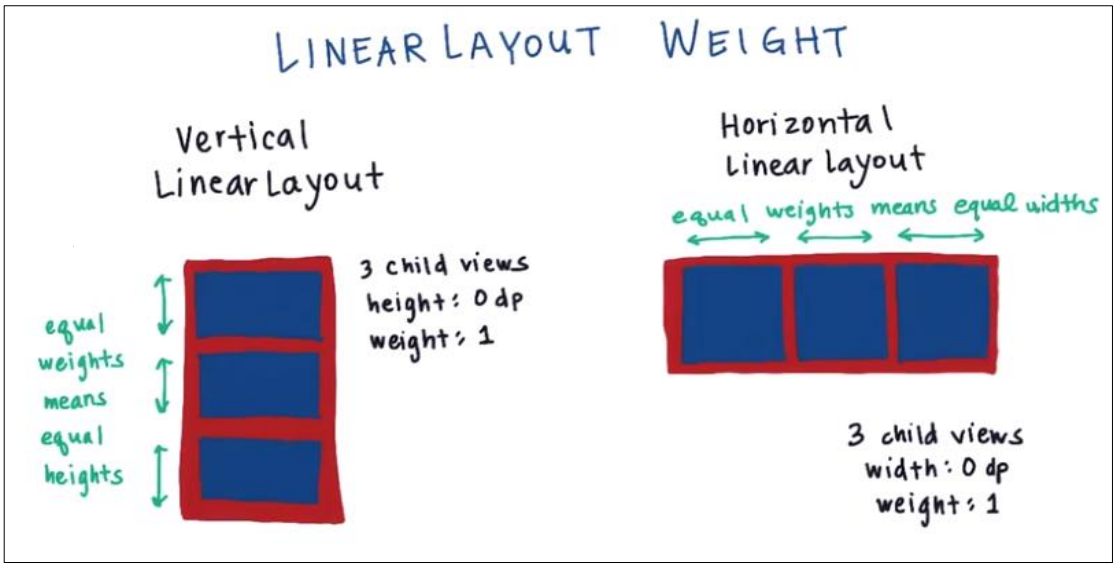
Now it's your time to go and make some changes with this quiz.



When designing an App for different devices or modes you might face a problem which shown in this example below.

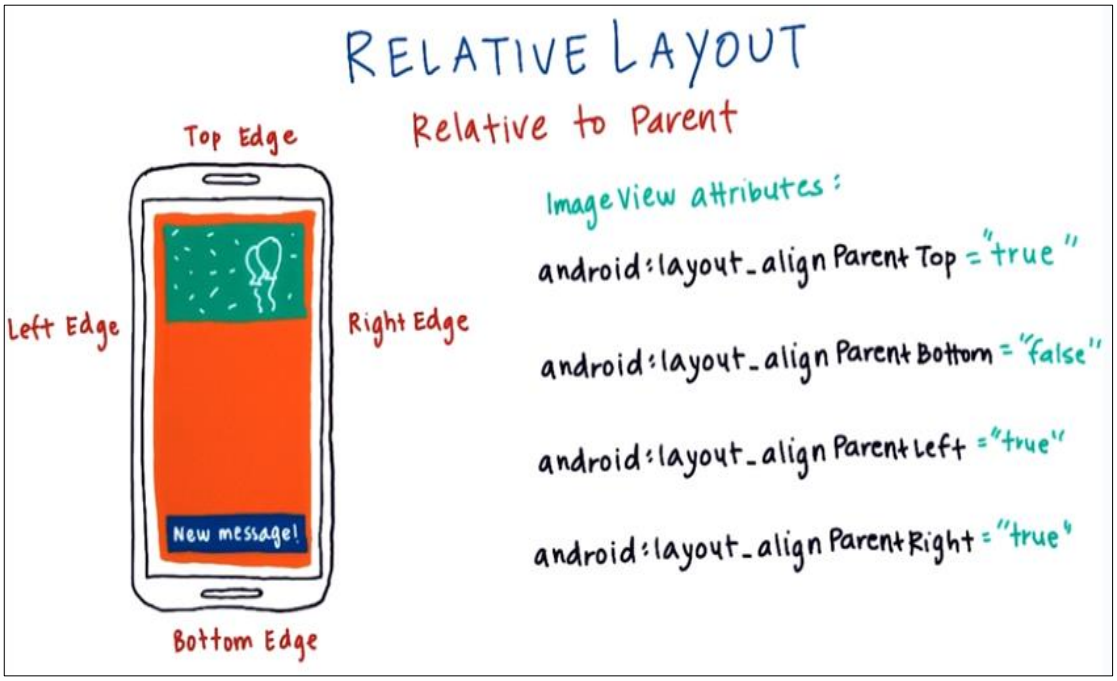


So to solve this problem we should use **layout_weight.** For horizontal layout go and set **layout_width** to **0dp** because we don't know how much space this view will take, then set the **layout_weight** for each of the buttons to **1**, the available width will be shared equally between the buttons. However, for vertical linear layout set **layout_height** to **0dp** and set the **layout_weight** for each of the buttons to **1**, the available width will be shared equally between the buttons. The figure below helps you to understand well.

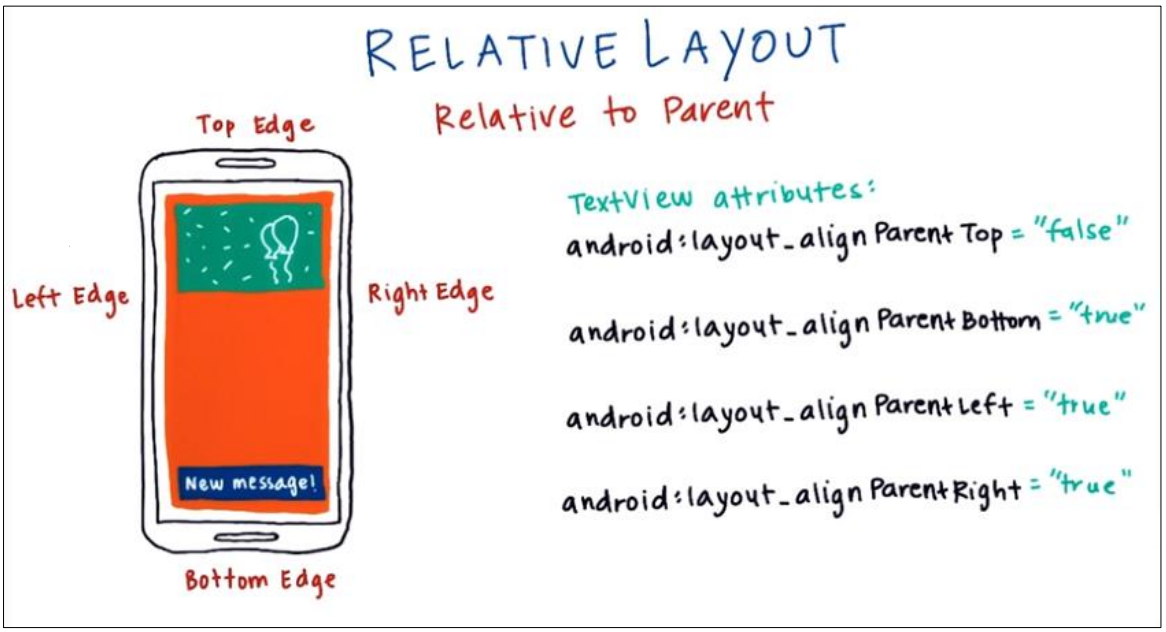


Relative Layout

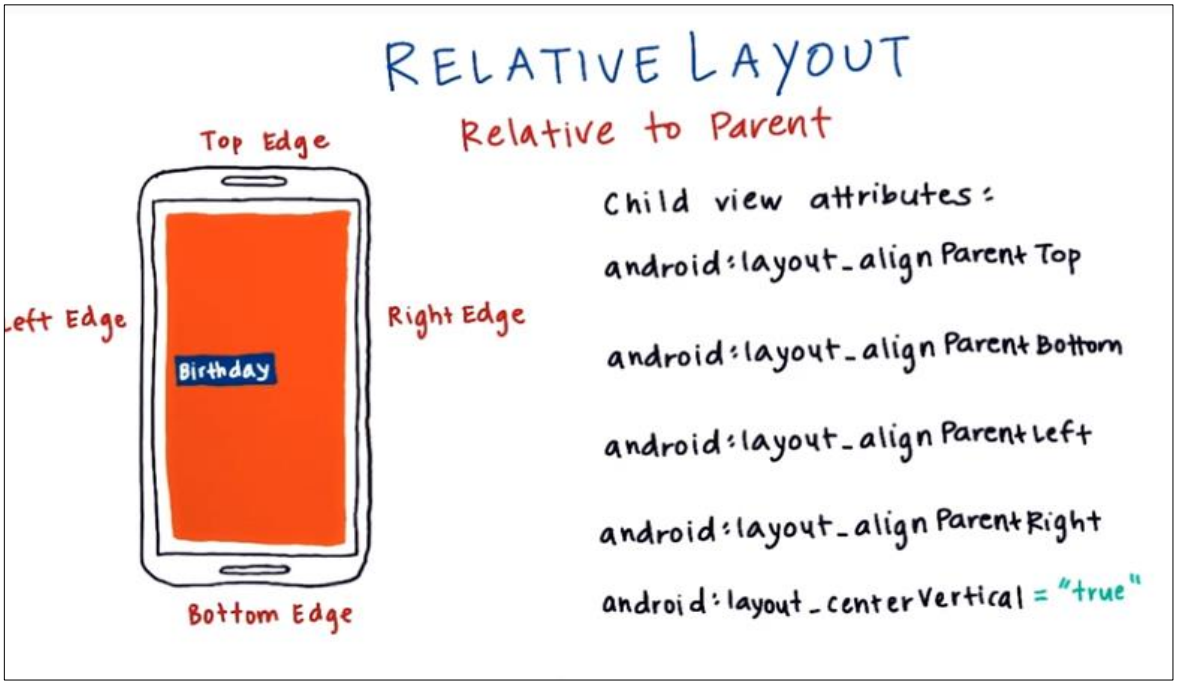
Here are two examples to make you get what is a relative layout, one for ImageView and another is for TextView.



Now take a look at the XML code for the TextView.

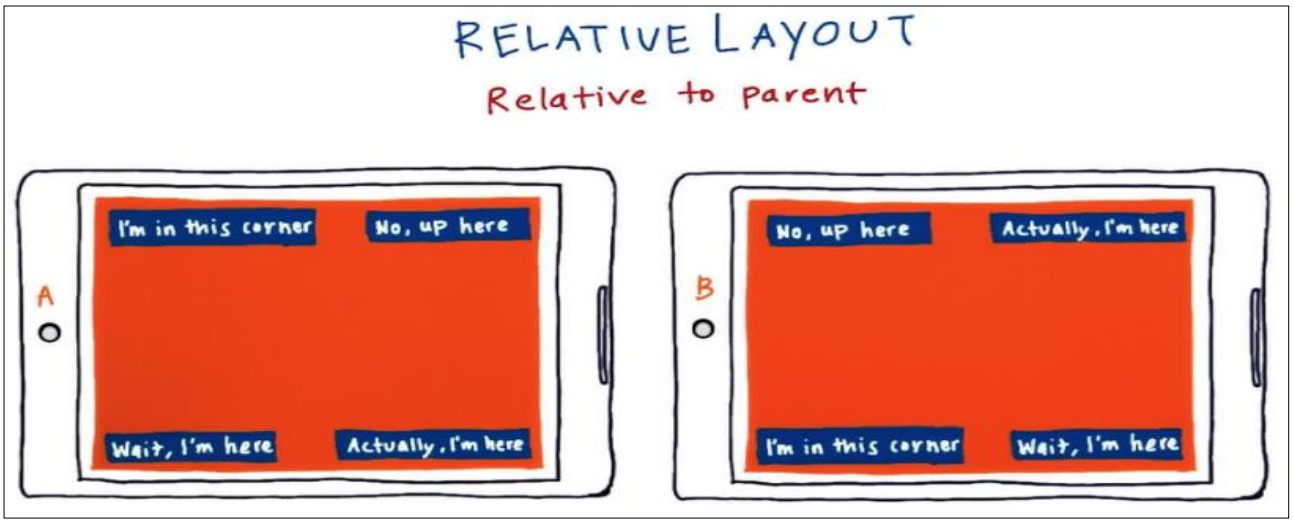


All of these attributes called view group layout parameters. Notice that all views added to relative layout are positioned at top left corner of the ViewGroup by default.

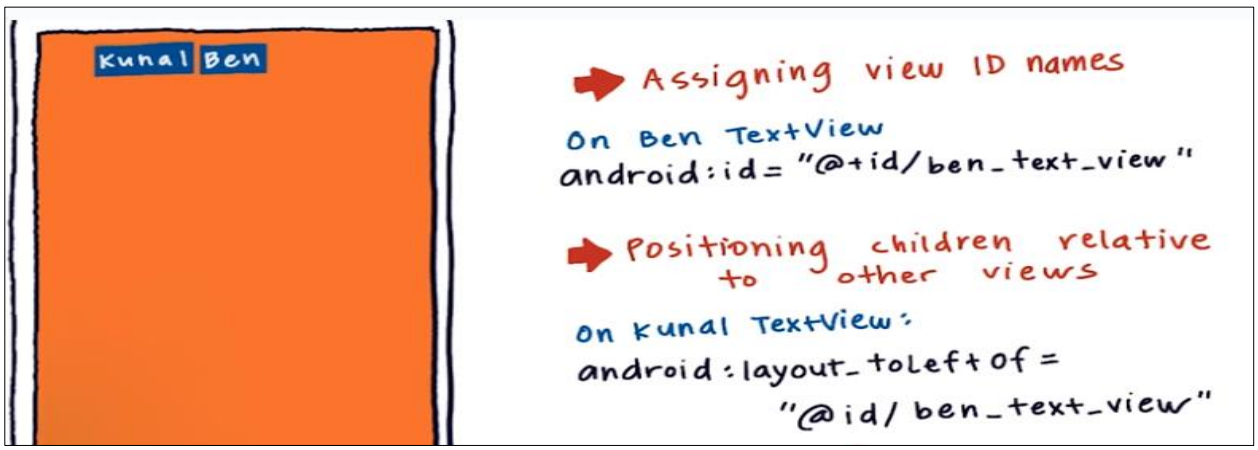


For this moment I want you to test yourself by doing this quiz, take a look at this **code**, then choose the correct result **A** or **B**.

```
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="16dp">

    <TextView
         android:text="I'm in this corner"
         android:layout_height="wrap_content"
         android:layout_width="wrap_content"
         android:layout_alignParentBottom="true"
         android:layout_alignParentLeft="true" />

    <TextView
        android:text="Wait, I'm here"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true" />

    <TextView
        android:text="No, up here"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true" />

    <TextView
         android:text="Actually, I'm here"
         android:layout_height="wrap_content"
         android:layout_width="wrap_content"
         android:layout_alignParentTop="true"
         android:layout_alignParentRight="true" />

    </RelativeLayout>
```



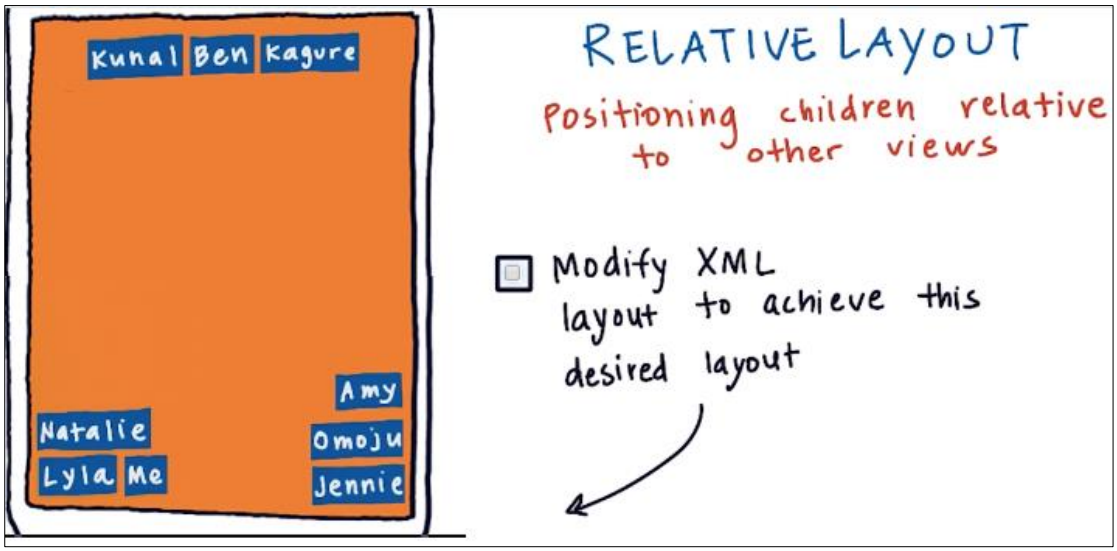
At this moment, we should also look at how to position views relative and this example will help you on this.



This is the code for the previous example.

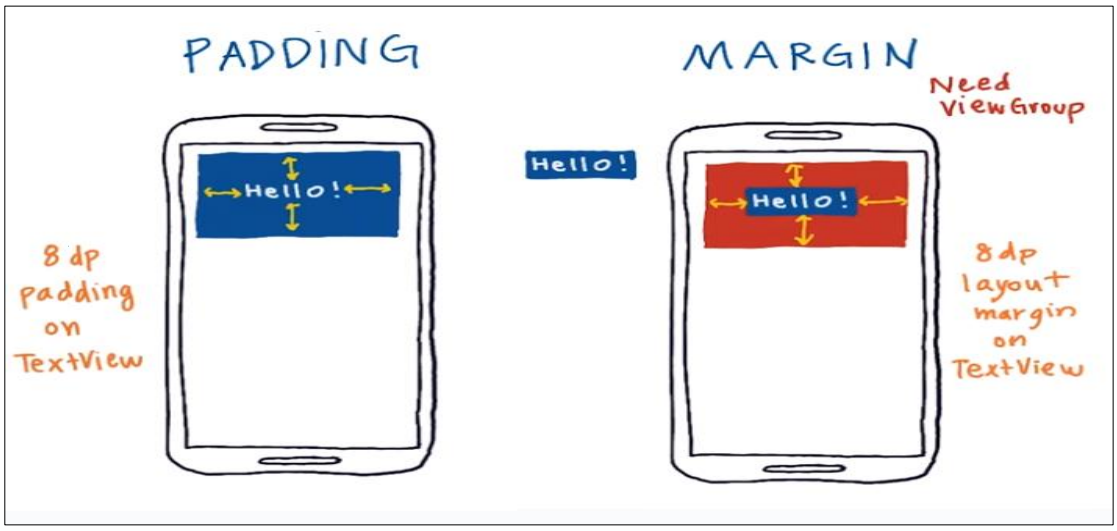
```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

<TextView
    android:id="@+id/ben_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="24sp"
    android:text="Ben" />

<TextView
    android:id="@+id/kunal_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_toLeftOf="@id/ben_text_view"
    android:textSize="24sp"
    android:text="Kunal" />
</RelativeLayout>
```

To make sure you can use positioning of relative layout views to each other, try this quiz.

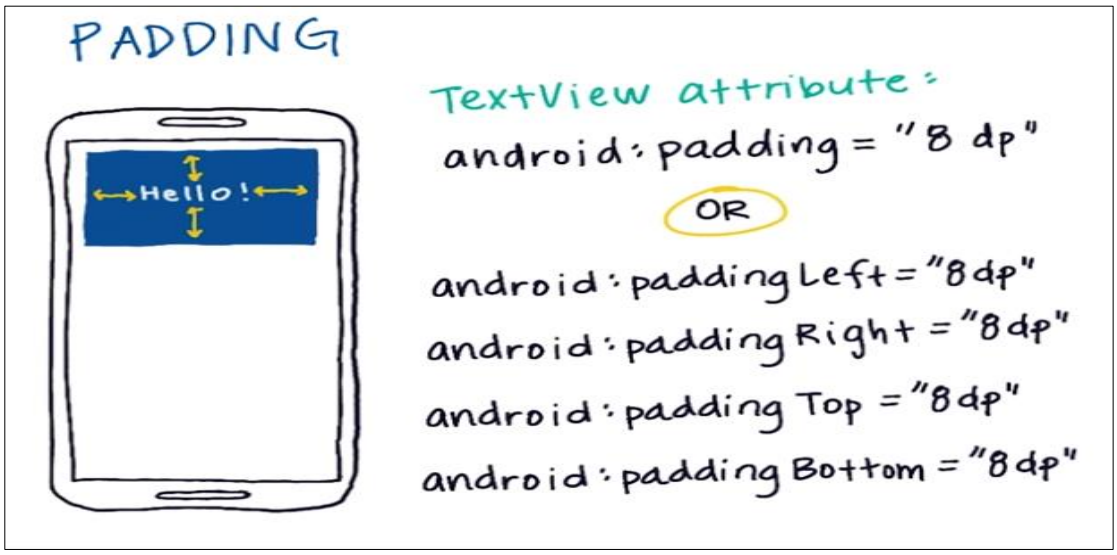


Padding and Margin

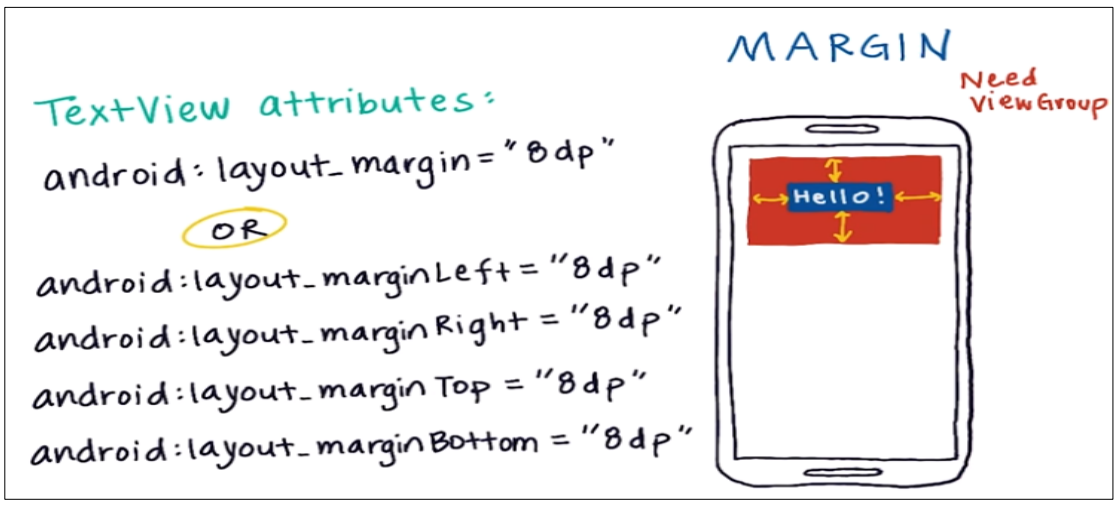
To control the white space between views or views and the edges of the screen you should know how to use padding and margins. This example shows the main difference.



To specify padding in XML, you take advantage of this instance.

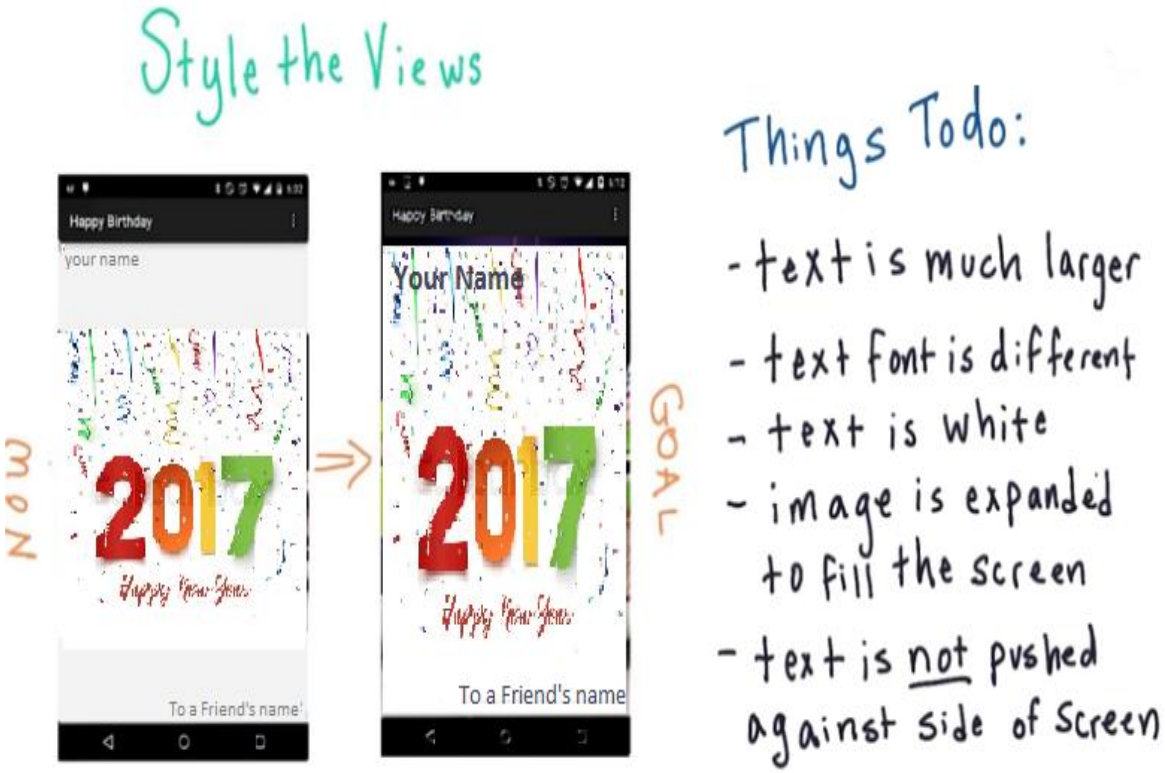


While to determine margins you may follow this figure.



Practice Set 1

It's your turn now to make your first App by doing these steps to achieve the goal shown below.

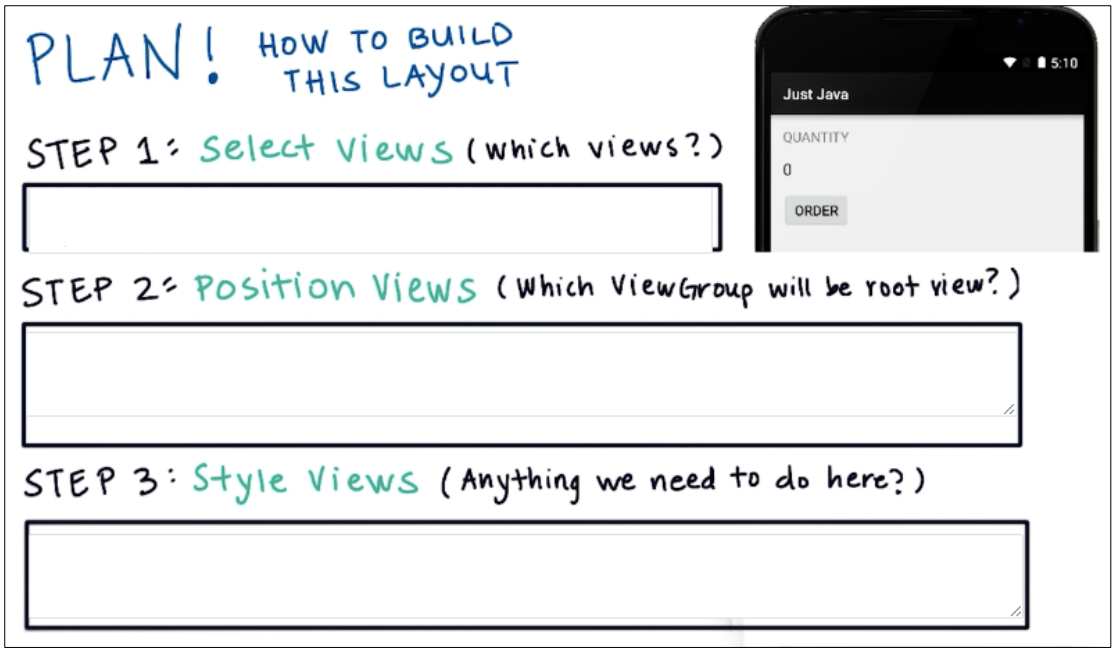


1- Change textSize to 36sp.
2- Change the font type to sans-serif-light by using fontFamily attribute.
3- Change the textColor to @anroid:Color/blue for both TextView.
4- Change layout_height, layout_width, and scaleType to expand the image.
5- If your text disappeared in **step 4**, just change the order of your views.
6- Make a space 20dp for TextViews on all sides push them from the corners of the screen.

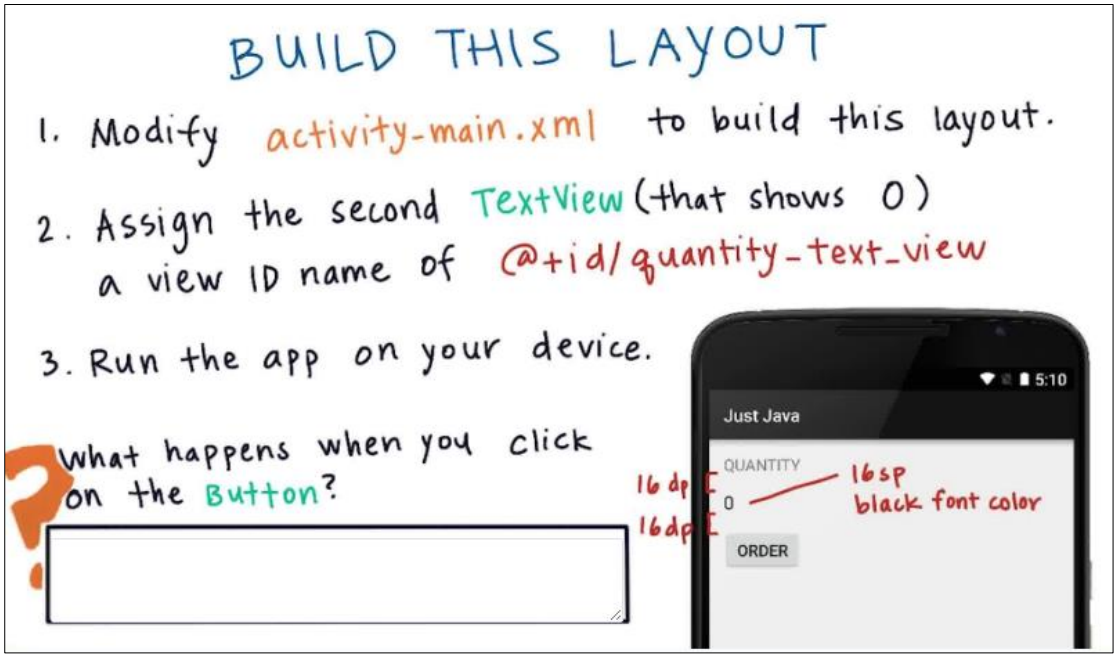
Score Keeper App:

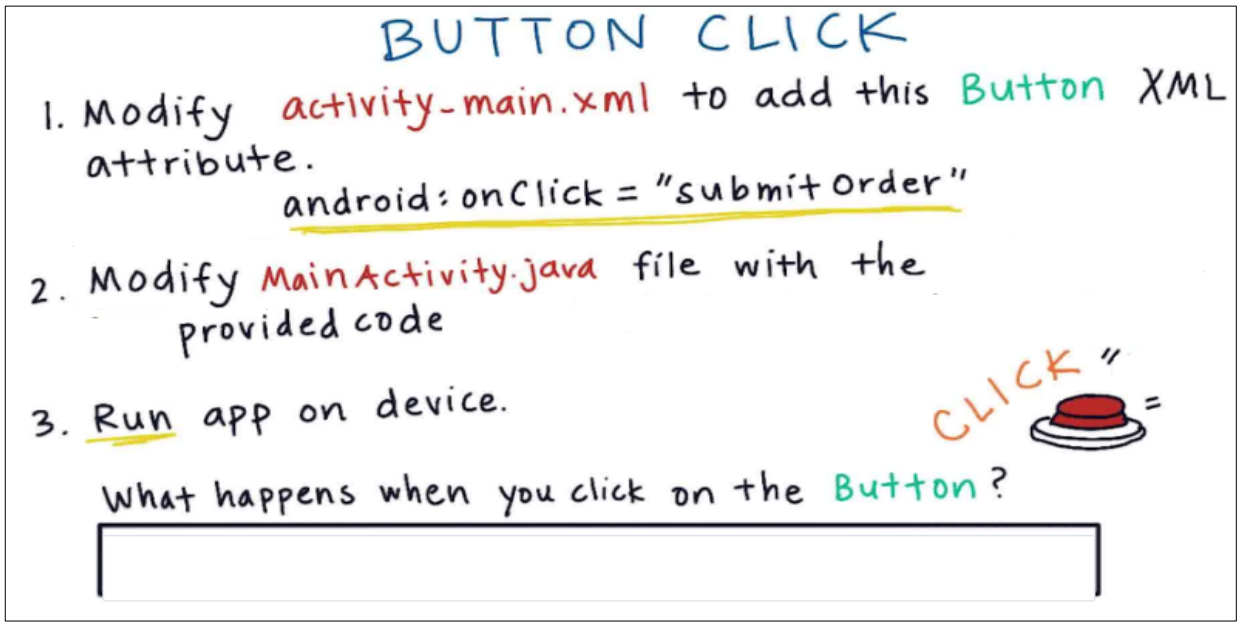
Making an App Interactive - **A**

For working on any project, before you start you need to plan on what requirement needed to accomplish your project. Next figure shows you an example.



After planning you start building the layout as you can see here.





Now put this code inside you **MainAcitivity** java file, inside **MainAcitivity** class and after **onCreate** method.

```
/**
 * This method is called when the order button is clicked.
 */
public void submitOrder(View view) {
    display(1);
}

/**
 * This method displays the given quantity value on the screen.
 */
private void display(int number) {
    TextView quantityTextView = (TextView) findViewById(R.id.quantity_text_view);
    quantityTextView.setText("" + number);
}
```

Then add this method after the **display** method, into Java **MainAcitivity** for displaying the price.

```
/**
 * This method displays the given price on the screen.
 */
private void displayPrice(int number) {
        TextView priceTextView = (TextView) findViewById(R.id.price_text_view);
        priceTextView.setText(NumberFormat.getCurrencyInstance().format(number));
}
```

You will probably get an error on **NumberFormat**, to fix it open Android studio and follow these steps.

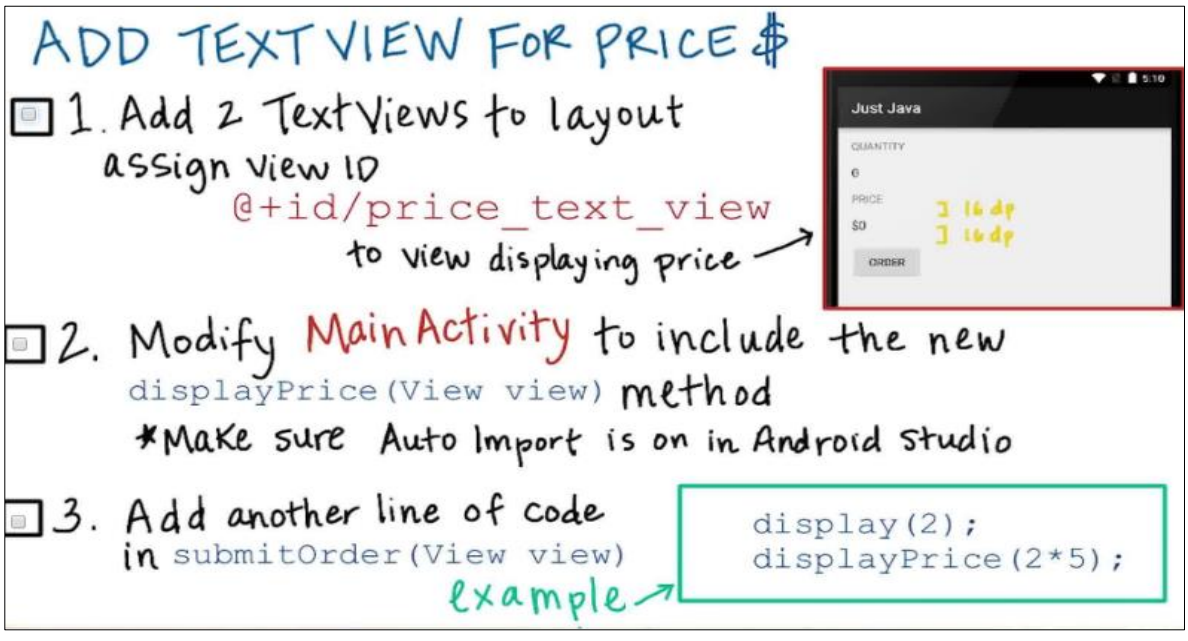
Make sure Auto Import is on:

- For Windows, go to File > Settings > Editor > General > Auto Import
- For Mac, go to Android Studio > Preferences > Editor > General > Auto Import

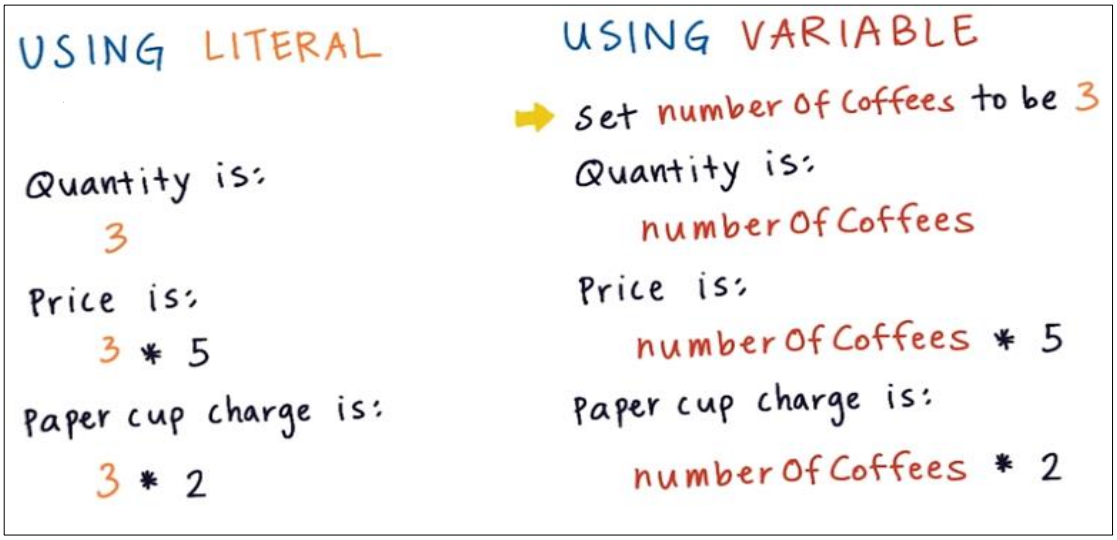
Check all of the following options:

- Show import popup
- Optimize imports on the fly
- Add unambiguous imports on the fly
- Insert imports on paste → All

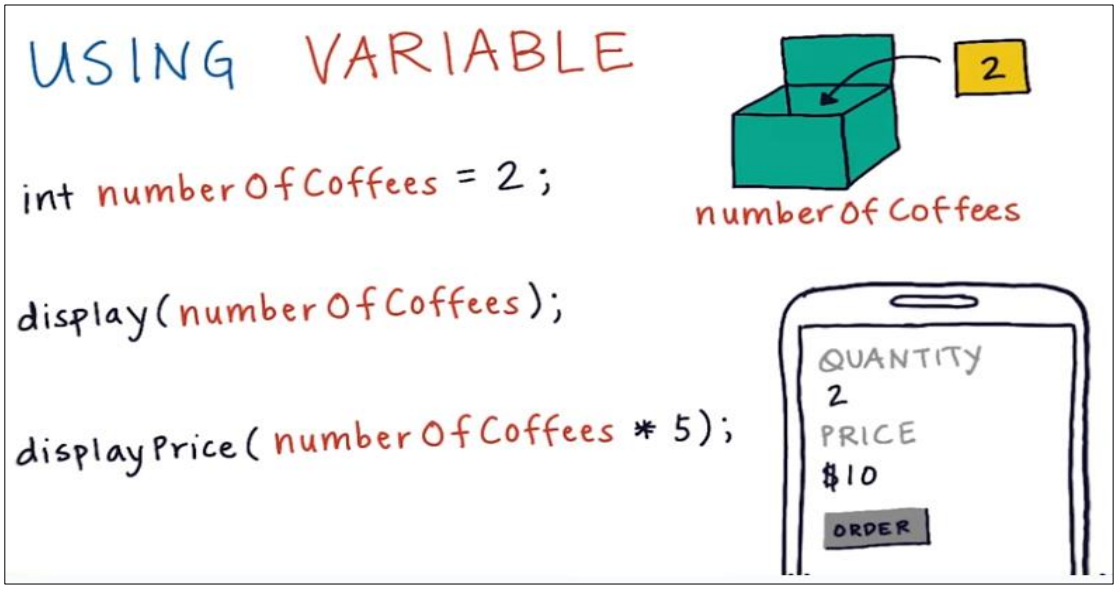
At this moment, check that you have done all of these changes.



Need for Variables



Let's use variables instead of literals, add the variable below to you code.



What happens if you change numberOfCoffees to 3! Go to your code and change the value.

Debugging a Crash

Even the most experienced mobile developers might face errors or their Apps crash during development, so it's important to know how to deal with errors /crashes. Try to make the following crash.

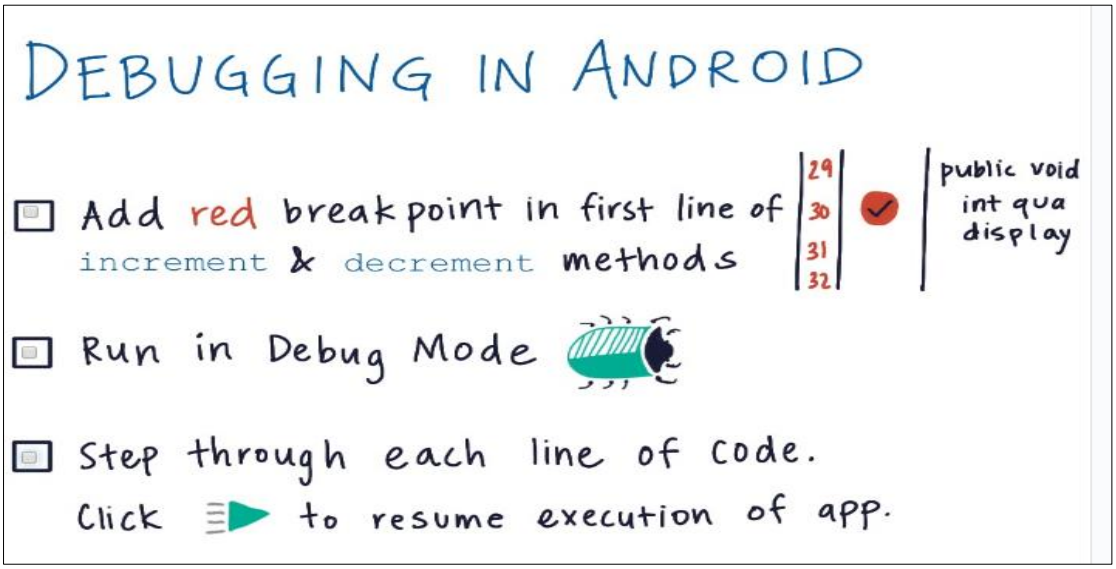
Hook Up Two Buttons

- [ ] 1. Modify activity_main
  - change layout
  - when + call increment
  - when - call decrement
- [ ] 2. For increment method
  - create quantity variable and initialize to 3
  - display quantity
- [ ] 3. For decrement method
  - create quantity variable and initialize to 1
  - display quantity

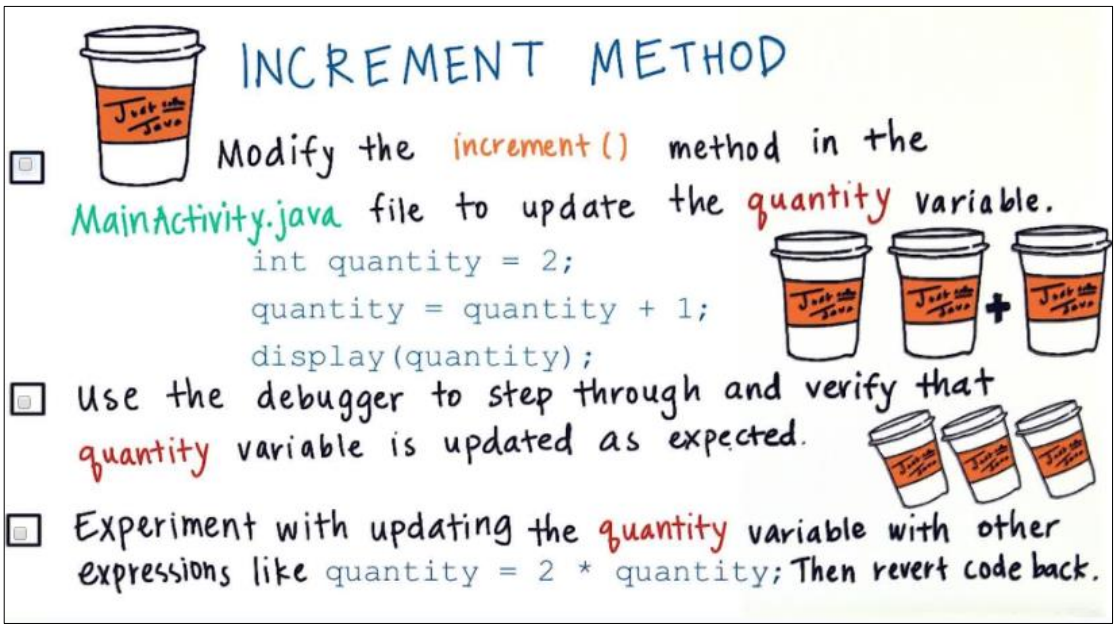
Debug Mode in Android Studio

Android Studio includes a debugger that allows you to debug apps running on the Android Emulator or a connected Android device. With the Android Studio debugger, you can:

- Select a device to debug your app on.
- Set breakpoints in your Java and C/C++ code.
- Examine variables and evaluate expressions at runtime.
- Capture screenshots and videos of your app.

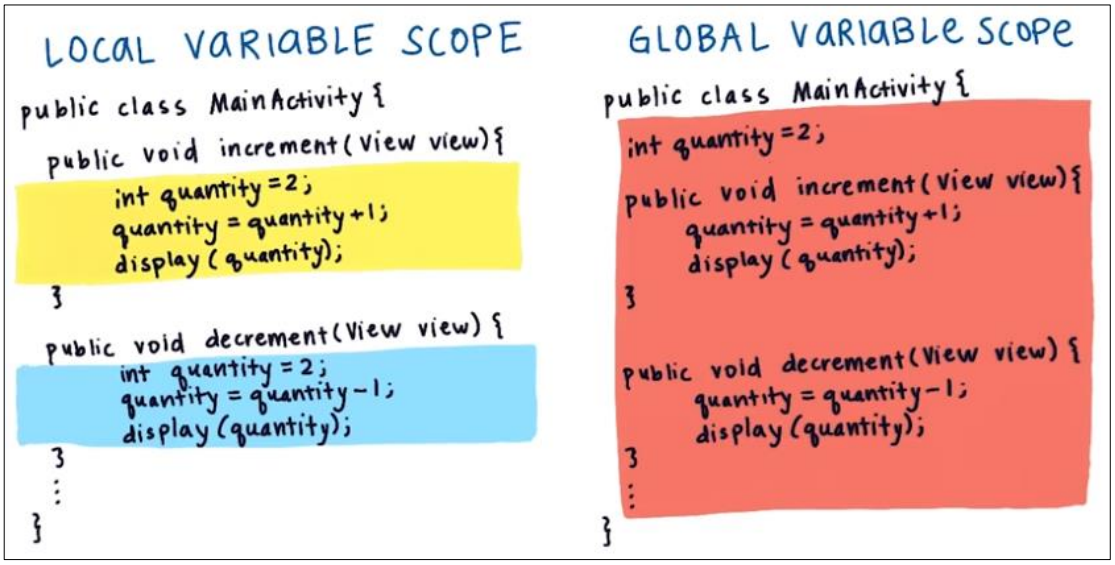


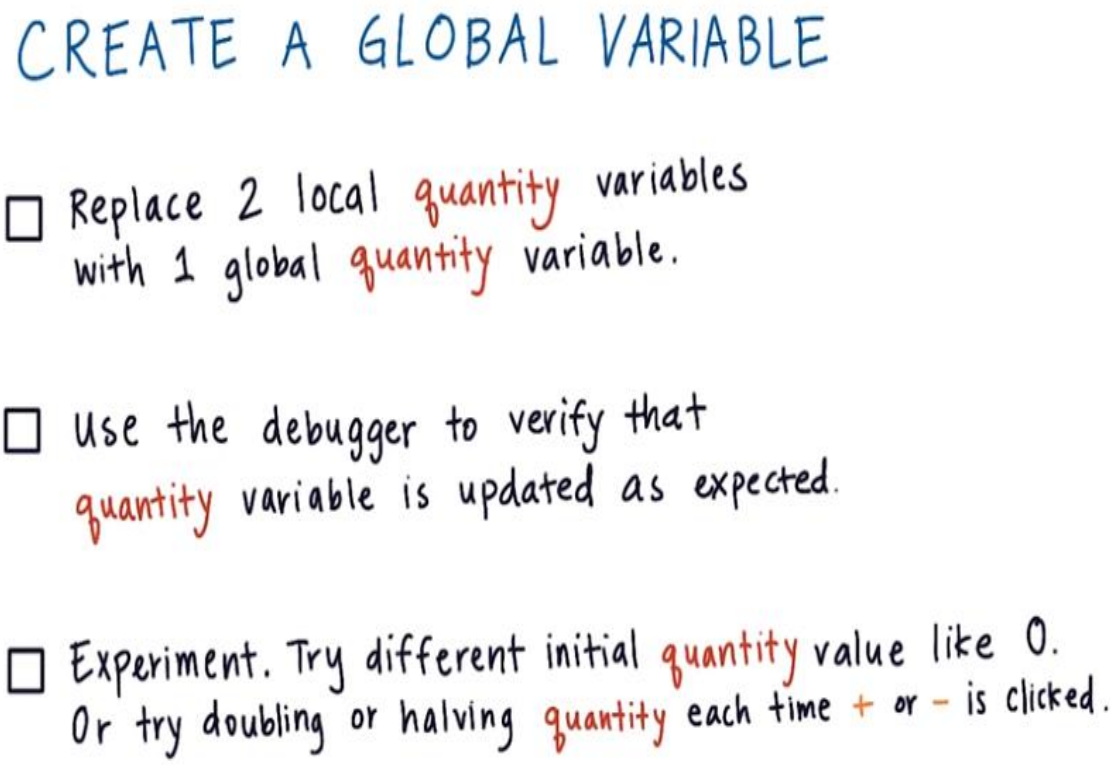
Modify the increment & decrement Methods



For decrement method, it's your turn to only do the **opposite** of increment method to make it works.

Make the Quantity Picker Work

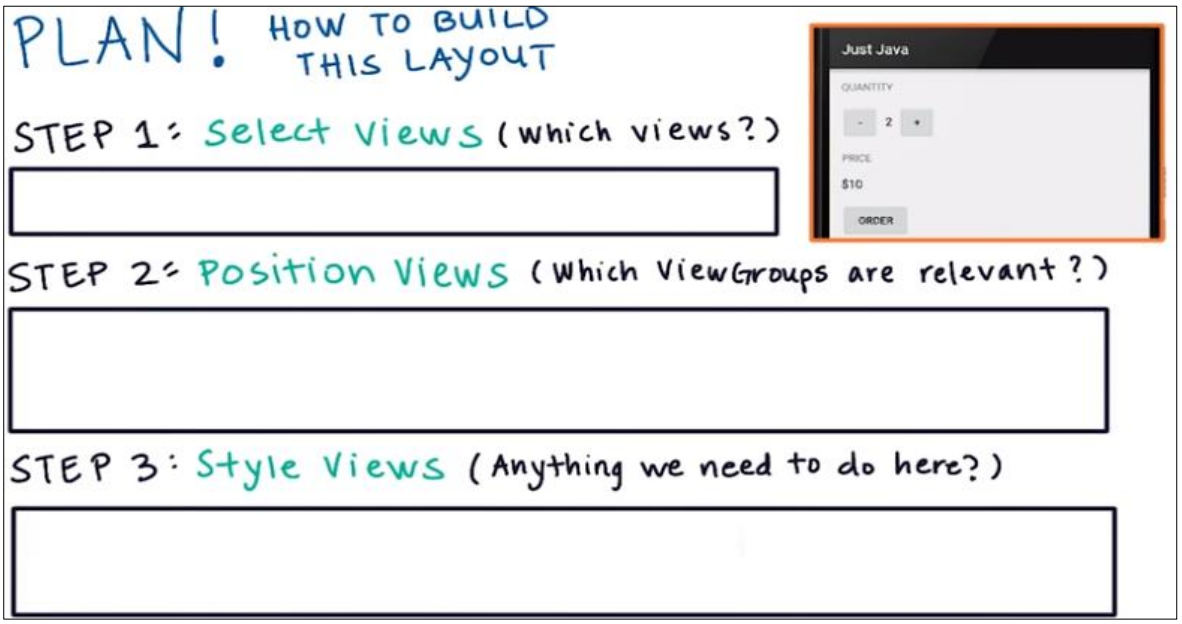




Score Keeper App:

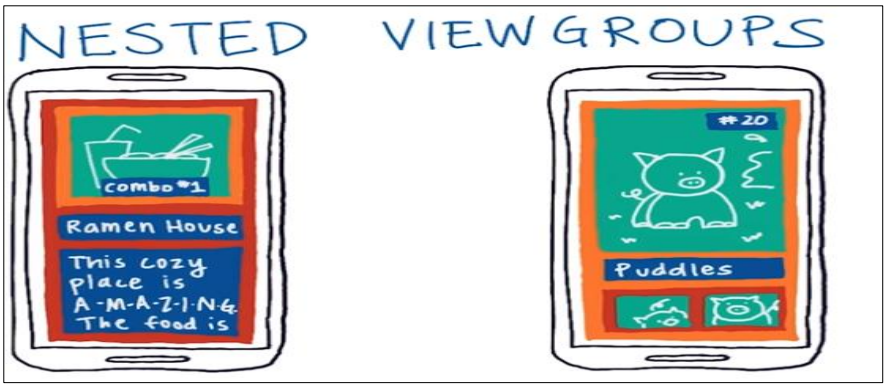
Making an App Interactive – **B**

We need to modify the quantity pickers buttons to be in horizontal in order to make better user experience by allowing user to scan the App faster, and to invest the area on the screen by not making all views on a narrow vertical area of the screen. Therefore, you should plan on what needed to fill the gap and change to the desired layout. Take some time to plan on building this layout.

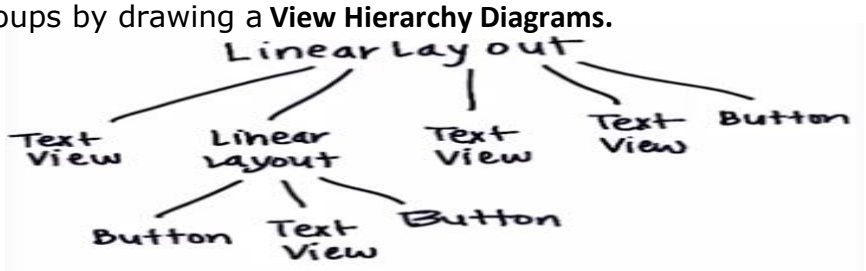


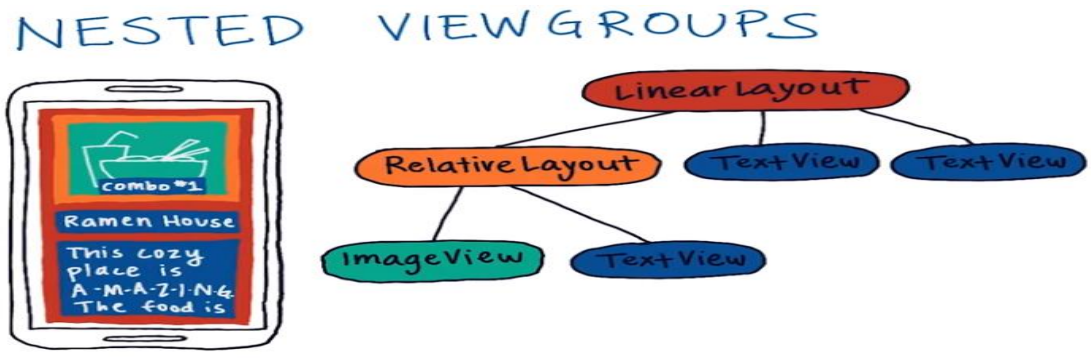
Nested ViewGroups

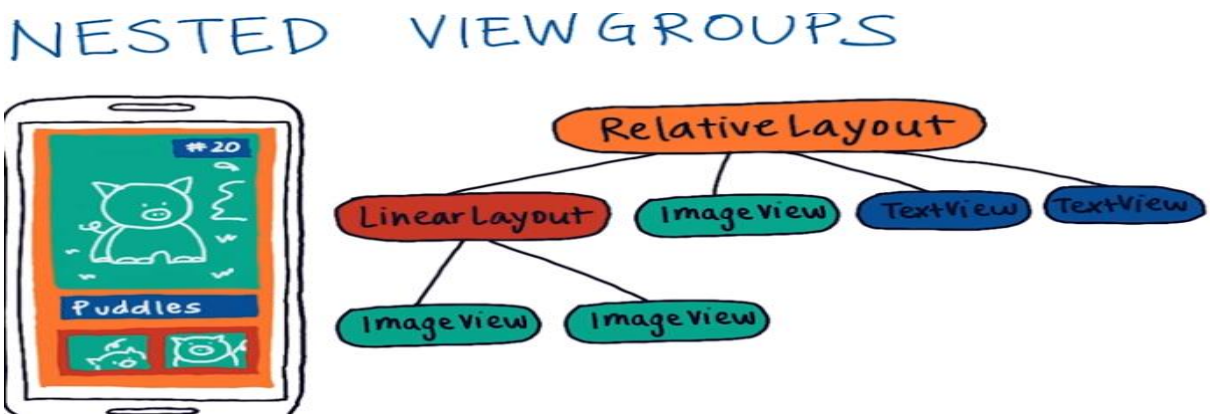
Nested ViewGroups mean putting ViewGroup inside another ViewGroups to build more complex and interested layouts like these.

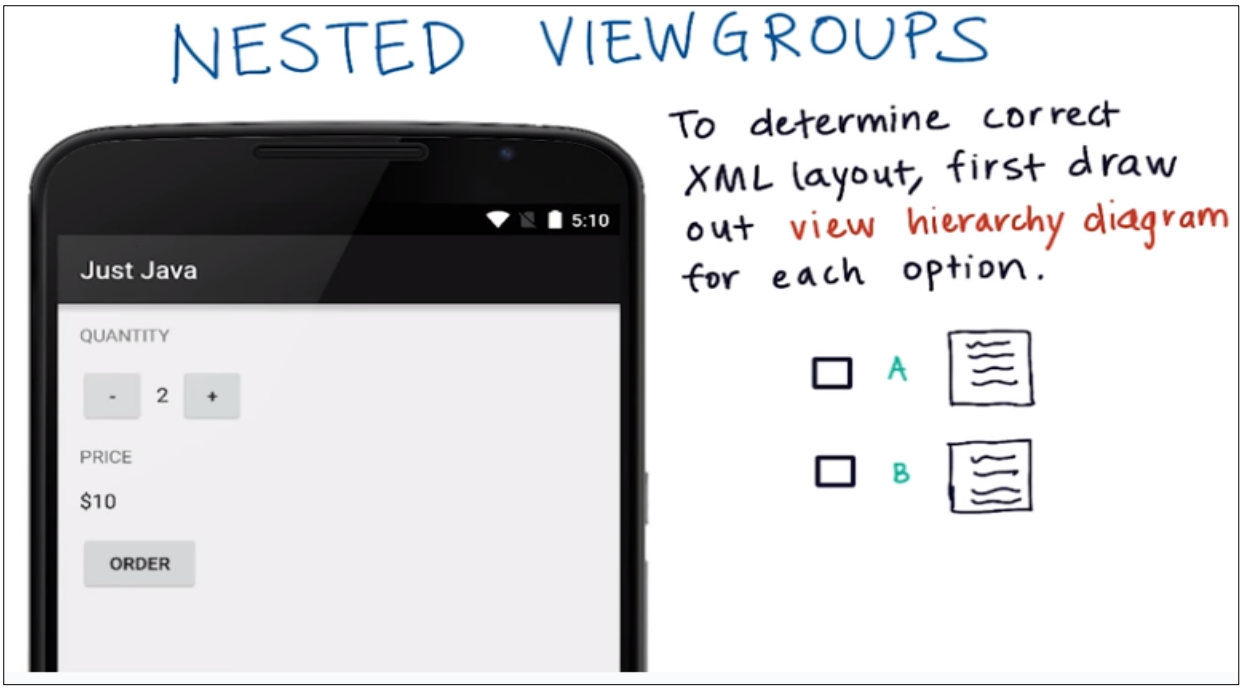


You can make nested layouts as much as you want to create complex layout, but be careful because relative ViewGroup need accurate positioning for its children views. Look at the example below and try to imagine the types of ViewGroups by drawing a **View Hierarchy Diagrams.**









Option A

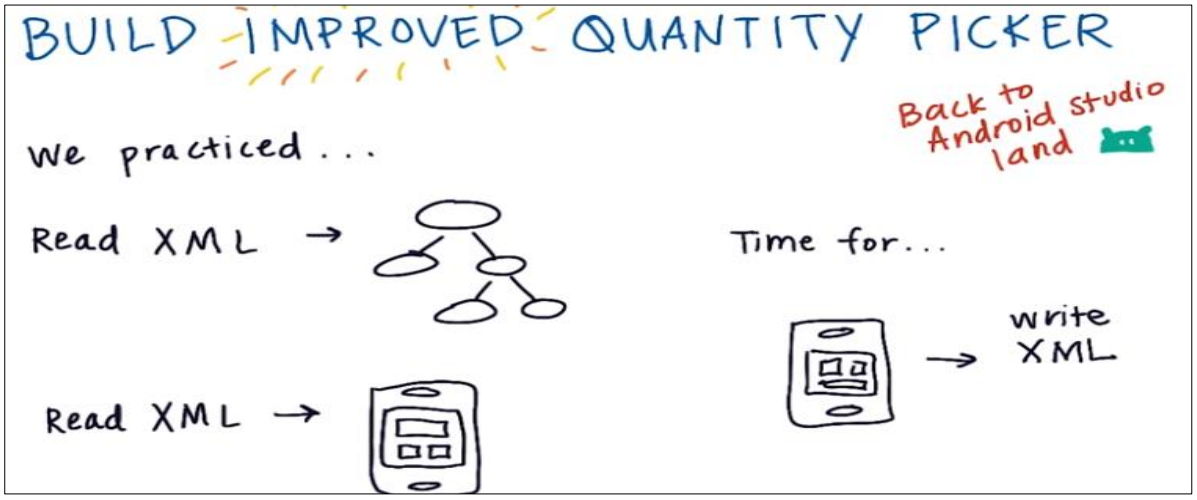
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginBottom="16dp"
      android:text="Quantity"
      android:textAllCaps="true" />

   <LinearLayout
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="horizontal">

      <Button
         android:layout_width="48dp"
         android:layout_height="48dp"
         android:onClick="decrement"
         android:text="-" />

      <TextView
         android:id="@+id/quantity_text_view"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_marginLeft="8dp"
         android:layout_marginRight="8dp"
         android:text="2"
         android:textColor="@android:color/black"
         android:textSize="16sp" />

      <Button
         android:layout_width="48dp"
         android:layout_height="48dp"
         android:onClick="increment"
         android:text="+" />

   </LinearLayout>

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_marginTop="16dp"
         android:text="Price"
         android:textAllCaps="true" />

      <TextView
         android:id="@+id/price_text_view"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_marginTop="16dp"
         android:text="$10"
         android:textColor="@android:color/black"
         android:textSize="16sp" />

      <Button
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_marginTop="16dp"
         android:onClick="submitOrder"
         android:text="Order" />

   </LinearLayout>
```

Option B

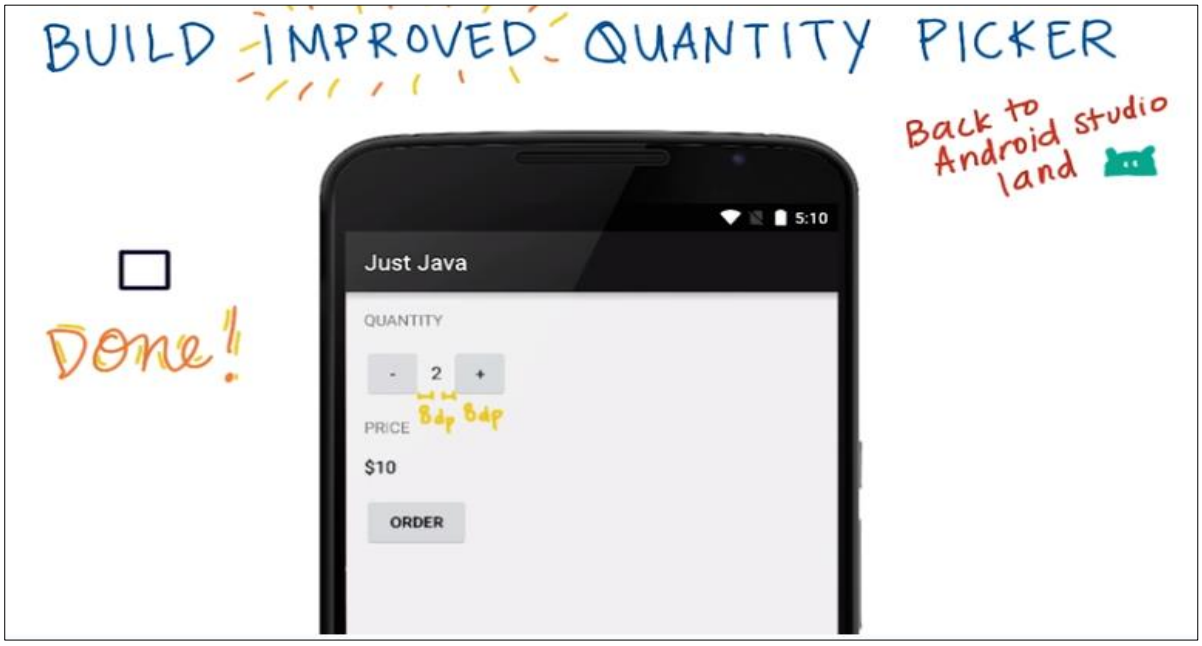
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginBottom="16dp"
      android:text="Quantity"
      android:textAllCaps="true" />

   <LinearLayout
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:orientation="vertical">

      <Button
         android:layout_width="48dp"
         android:layout_height="48dp"
         android:onClick="decrement"
         android:text="-" />

      <TextView
         android:id="@+id/quantity_text_view"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_marginLeft="8dp"
         android:layout_marginRight="8dp"
         android:text="2"
         android:textColor="@android:color/black"
         android:textSize="16sp" />

      <Button
         android:layout_width="48dp"
         android:layout_height="48dp"
         android:onClick="increment"
         android:text="+" />

   </LinearLayout>

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="16dp"
      android:text="Price"
      android:textAllCaps="true" />

   <TextView
      android:id="@+id/price_text_view"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="16dp"
      android:text="$10"
      android:textColor="@android:color/black"
      android:textSize="16sp" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="16dp"
      android:onClick="submitOrder"
      android:text="Order" />

</LinearLayout>
```

Build the Quantity Picker Layout

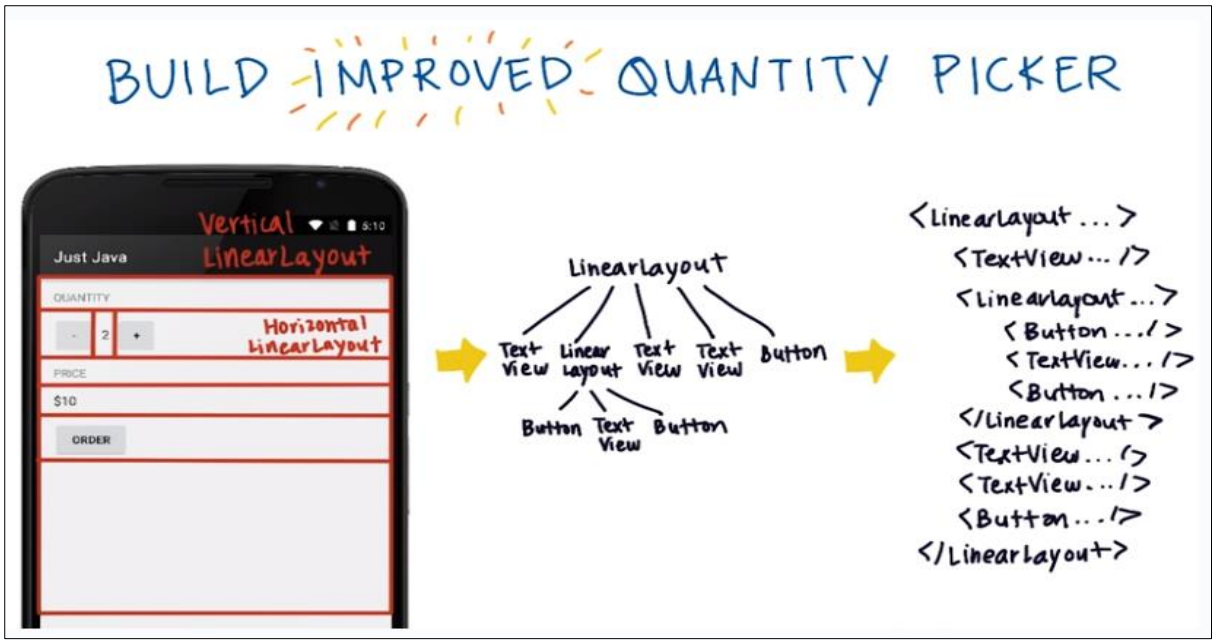
It's very necessary concept to know nested layouts because almost all screens contain some type of nested layouts. Now I would like that you go to Android Studio to implement changes for an improved quantity picker.



Go to the activity_main.xml file and make the required changes to get the same result below.



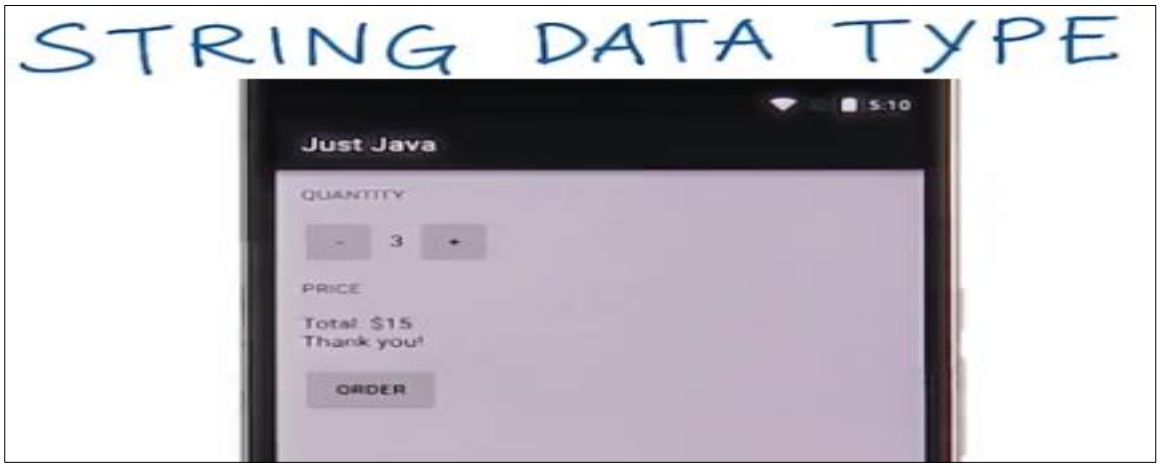
If you need a help on how to this task, check the diagram below so you know how to plan and build this layout.



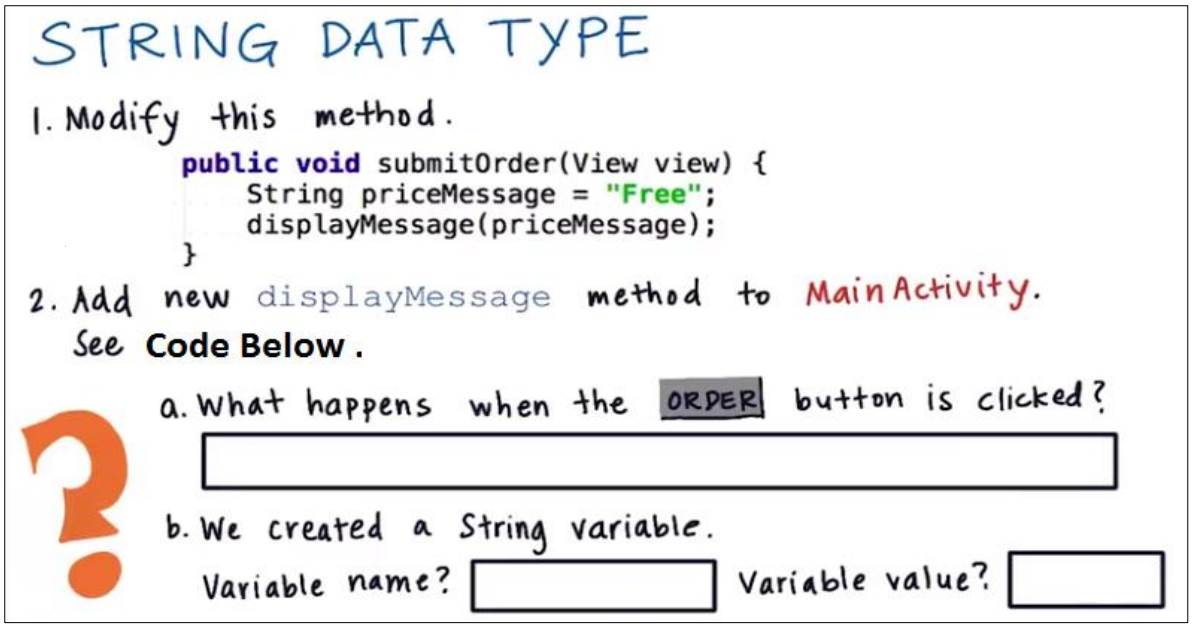
Now I think your code has a bunch of extra white spaces and the ordering of attributes may be incorrect according to the Android code style guidelines, so in the menu at the top of your screen click on **code** then click on **reformat code** and also **rearrange code.** Let's run the app.

String Data Type

Currently the price is integer and we want to make it string to print number and text such as the "total is 15 $".



May be you wonder why we just use XML to put the text, but this will make the text static and may not be related to the number. A good example when a person calls you if we xml is used then any person who calls you the same name you will see regardless the number who is calling.

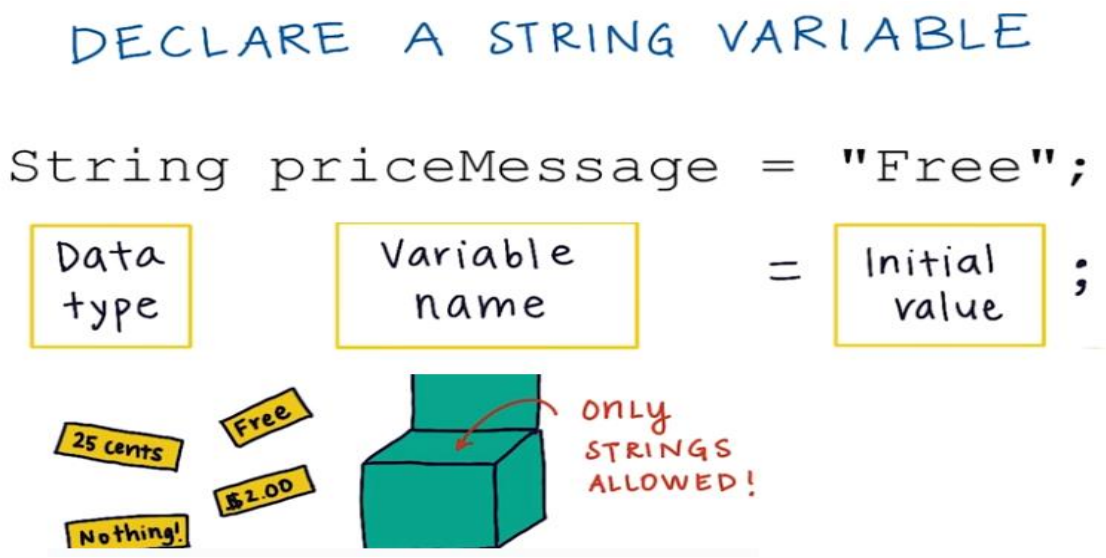


Add this method before the closing brace of MainAcitivity.

New method in MainActivity.java

```
/**
 * This method displays the given text on the screen.
 */
private void displayMessage(String message) {
    TextView priceTextView = (TextView) findViewById(R.id.price_text_view);
    priceTextView.setText(message);
}
```

Create the String Variable



Naming of variable

According to Oracle.com: Every programming language has its own set of rules and conventions for the kinds of The rules and conventions for naming your variables in **Java** can be summarized as follows:

- Variable names are case-sensitive. A variable's name can be any legal identifier — an unlimited-length sequence of Unicode letters and digits, beginning with a letter, the dollar sign "$", or the underscore character "_". The convention, however, is to always begin your variable names with a letter, not "$" or "_". Additionally, the dollar sign character, by convention, is never used at all. You may find some situations where auto-generated names will contain the dollar sign, but your variable names should always avoid using it. A similar convention exists for the underscore character; while it's technically legal to begin your variable's name with "_", this practice is discouraged. White space is not permitted.
- Subsequent characters may be letters, digits, dollar signs, or underscore characters. Conventions (and common sense) apply to this rule as well. When choosing a name for your variables, use full words instead of cryptic abbreviations. Doing so will make your code easier to read and understand. In many cases it will also make your code self-documenting; fields namedcadence, speed, and gear, for example, are much more intuitive than abbreviated versions, such as s, c, and g. Also keep in mind that the name you choose must not be a keyword or reserved word.
- If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalize the first letter of each subsequent word. The names gearRatio and currentGear are prime examples of this convention. If your variable stores a constant value, such as static final int NUM_GEARS = 6, the convention changes slightly, capitalizing every letter and separating subsequent words with the underscore character. By convention, the underscore character is never used elsewhere.

Escape Sequences

A character preceded by a backslash (\) is an *escape sequence* and has special meaning to the compiler. The following table shows the Java escape sequences:

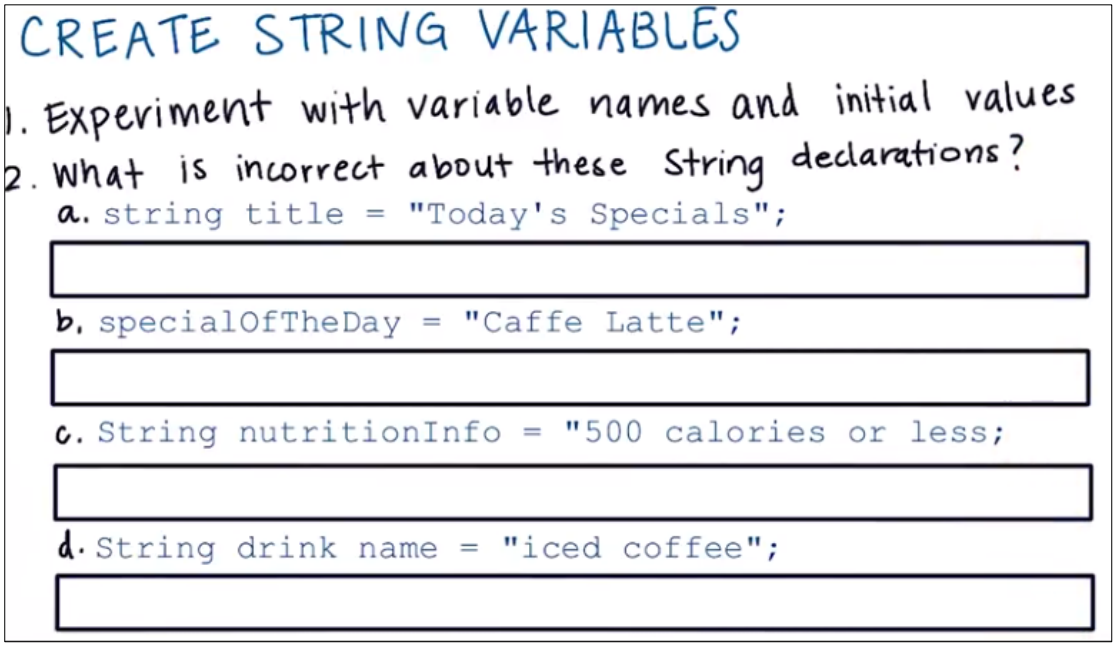
Escape Sequences

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes. To print the sentence `She said "Hello!" to me.`

you would write `System.out.println("She said \"Hello!\" to me.");`

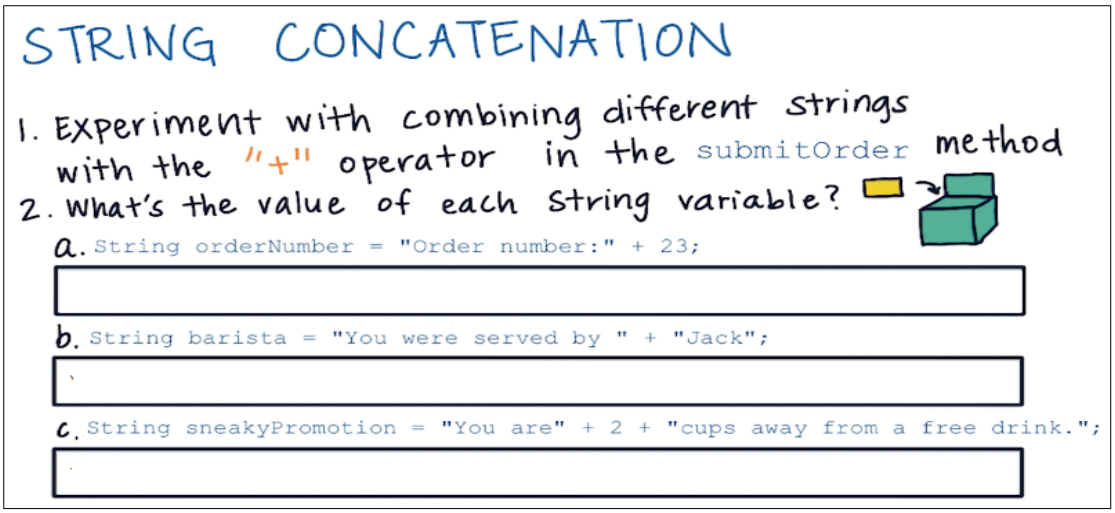
Now in this Quiz, I ask you to determine the errors or bugs in these lines of codes.



Combining Strings Together

When dealing with string in Java, an important term to understand is string concatenation, it means we are joining character string together end to end. To concatenate these strings together we use the plus "**+**" operator. Example below shows 3 strings concatenation.

There's no extra spaces among these strings, if you need to add space you must add space at the end of of the first string, or at the end or beginning of the second and third strings literals. Note that spaces around + symbol don't contribute to the display of string. If you concatenate a string with an integer then whole thing will turn to a string. According to Android code style guidelines we should have a while space before and after operators such as + symbol. Now its your time to exercise with string concatenations.

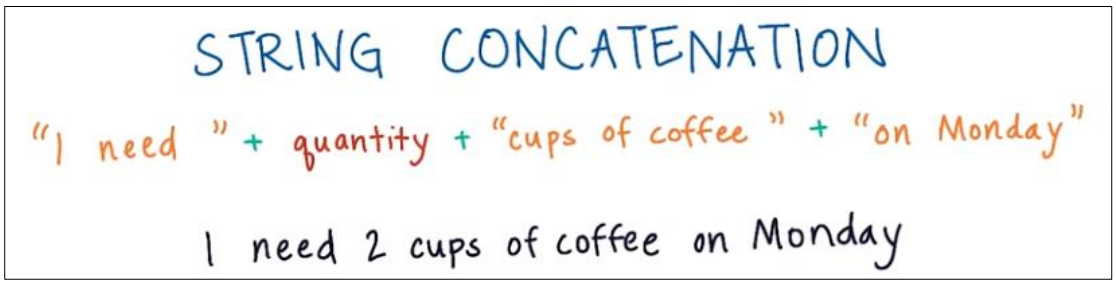


Go to your code and, Start by modifying submitOrder to say:

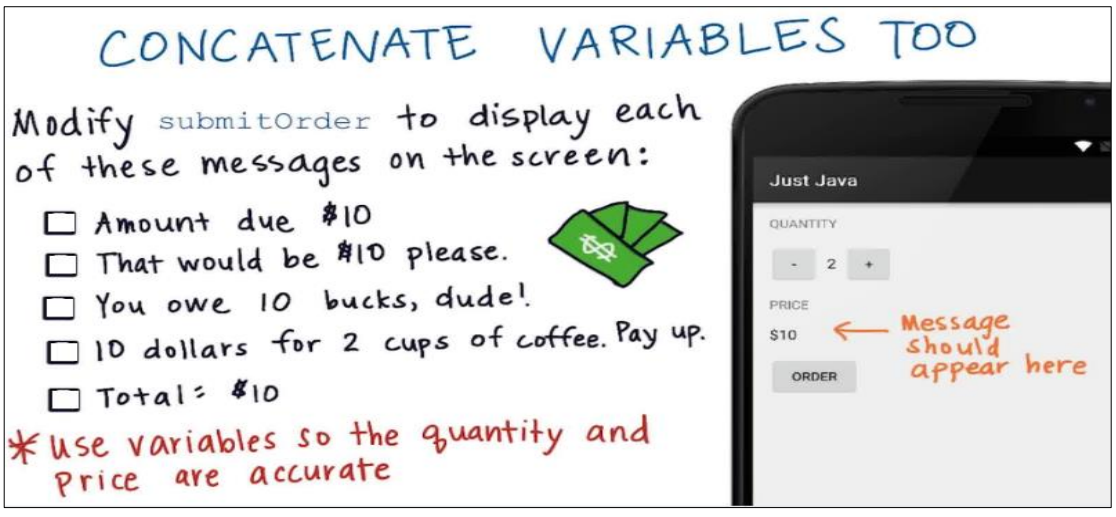
```
String message = "Item count " + quantity;
displayMessage(message);
```

Combining String and Integer Variables

We will change the **literal** value of **2** in the previous example, and we make it a **variable**.

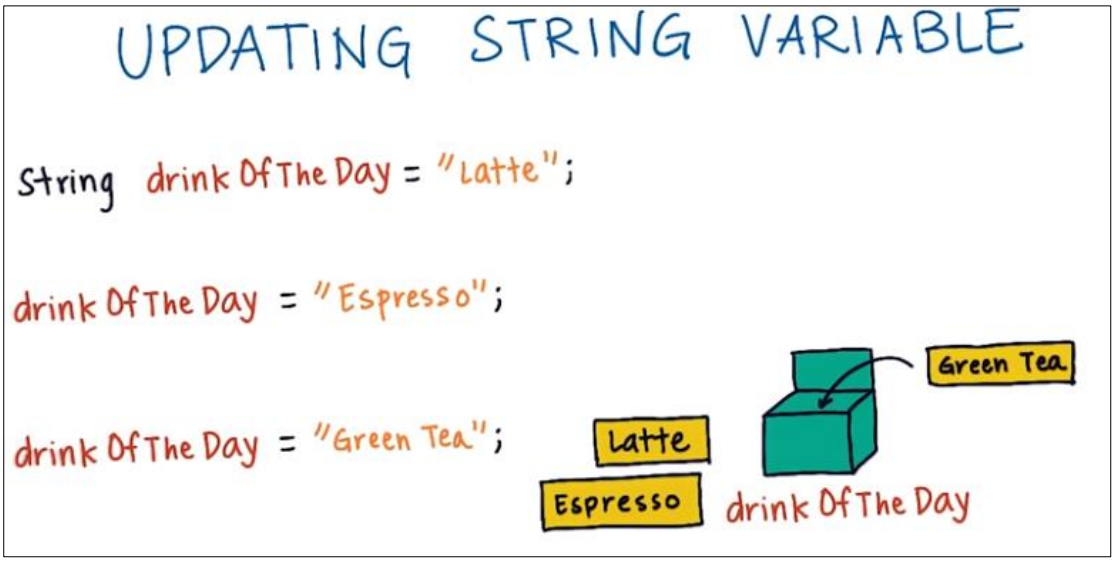


The variable will be always updated to latest value, while the other parts will remain constant and they don't change.

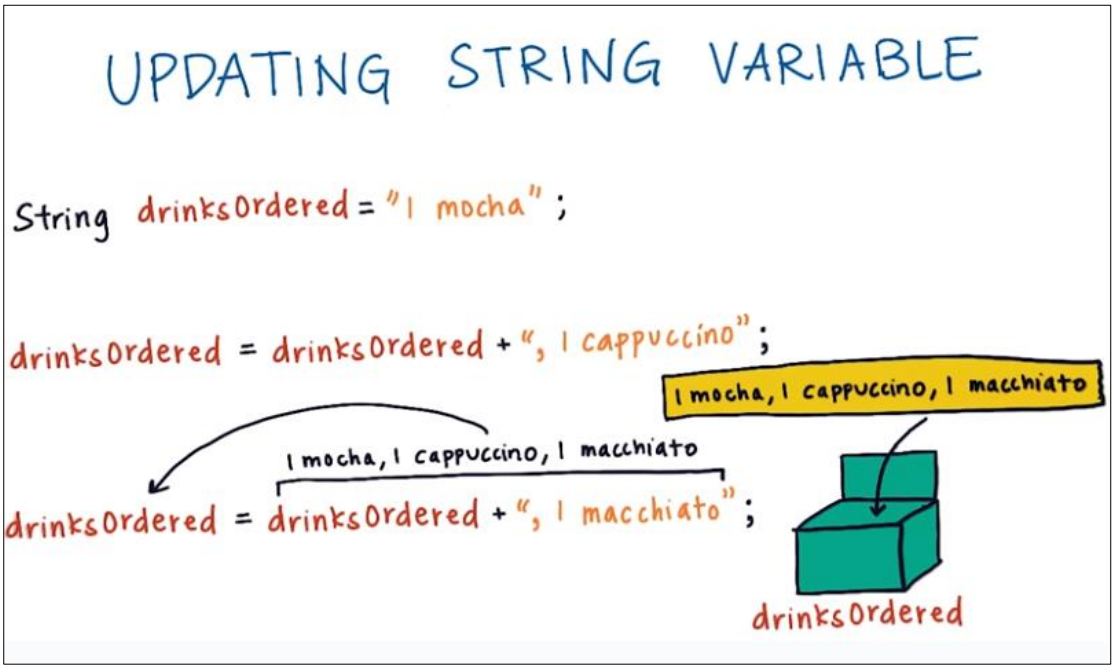


Update the String Variable

We can use string variable to display the drink of the day for example. The variable is start with "Latte", on next day we just update the string variable to say "Espresso", and then on other day we can update to "Green Tea".

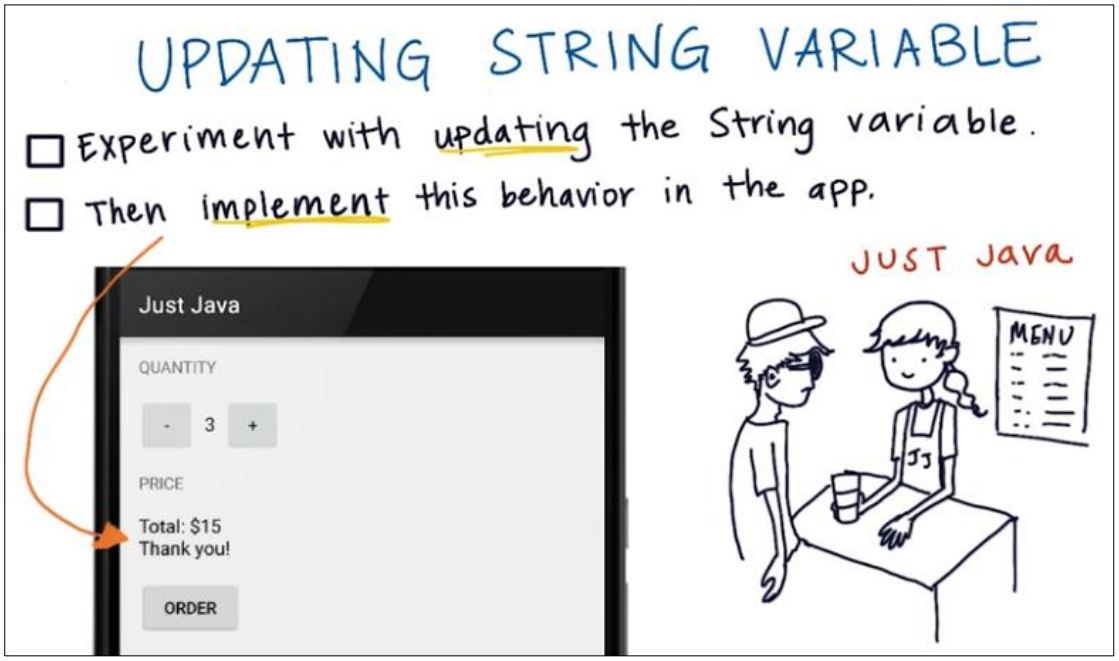


Here is another example below.




```
/**
 * This method is called when the order button is clicked.
 */
public void submitOrder(View view) {
    int price = quantity * 5;
    String priceMessage = "Total: $" + price;
    priceMessage = priceMessage + " Woohoo!";
    displayMessage(priceMessage);
}
```

At this moment you should be familiar with knowing how to update variables in Android, then move to Android studio and do the following tasks:



When you change the quantity and hit the ORDER button, it should display the Total, Price, and Thank you!. In the next lesson you'll need to update string variables more often, because there will be more fields in this form, and we are going to need to build up the order summary.

How to Learn More on Your Own

Find an interesting Android article, have you ever had a class where the instructions are to browse the internet for whatever interests you? Well, here it is. Search online to find another Android development article that you're interested in and read it (or it can be a video or podcast). Figure out what the key ideas of the piece are and share those ideas with others. You can start by checking out these sites.

- **Android Weekly** is a weekly newsletter about the latest news in Android development.
- **MaterialUp** curates the best of Material design inspiration and tools.

Build up your network of Android developers on social media

Google Developer Experts are a network of professional developers in industry (external to Google). They are recognized as leaders in their communities - speaking at conferences, publishing tutorials, and

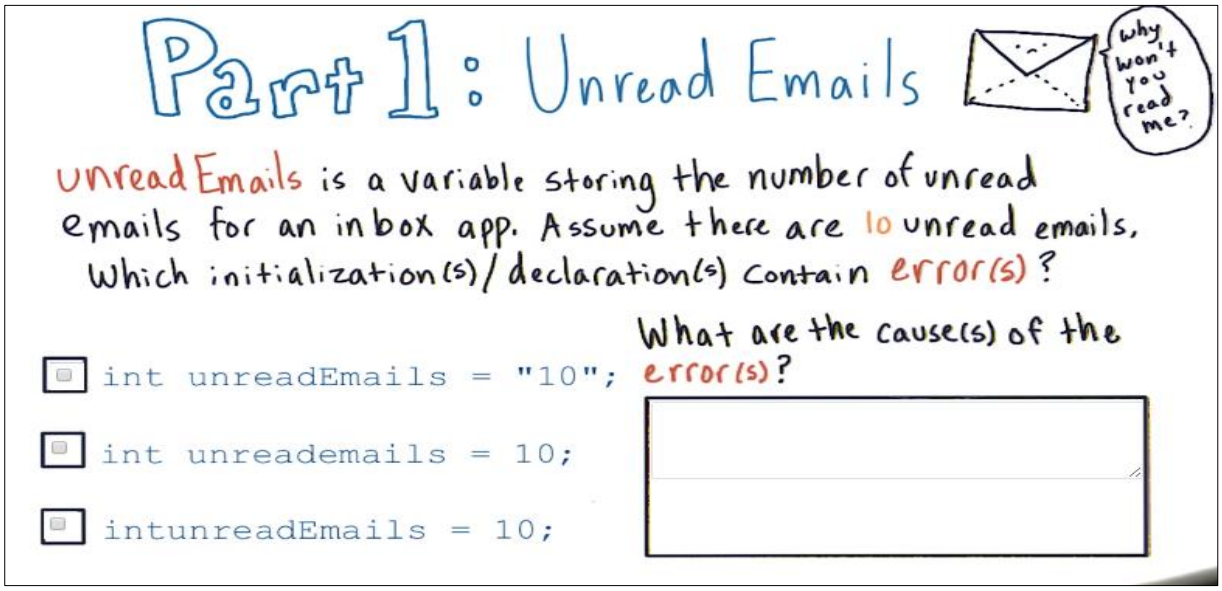
mentoring developers. You can look at their profiles here and follow them on social media.
There are also multiple communities that you can be involved with. These are great places to ask those burning Android questions.

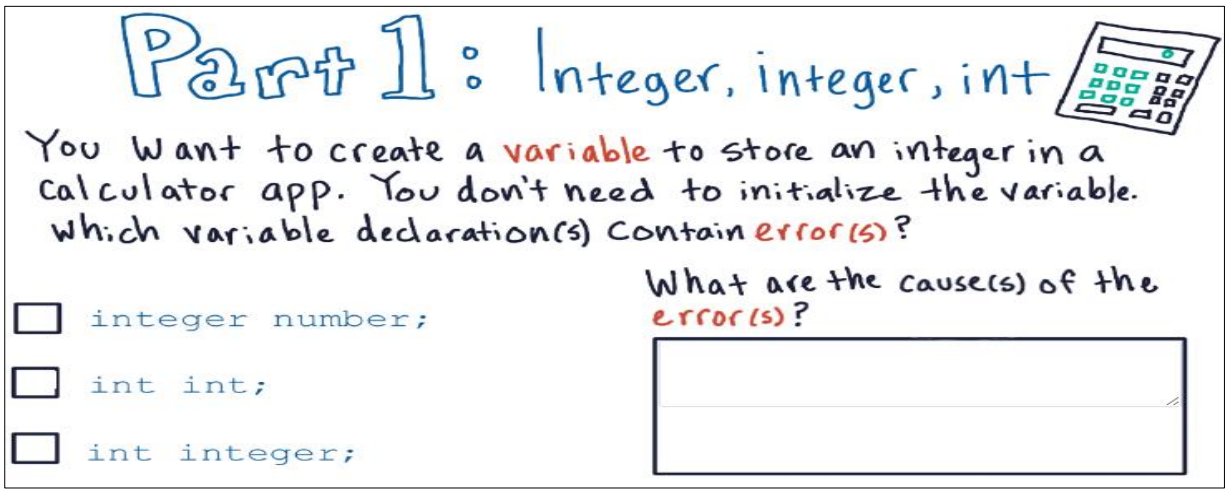
- Android Development G+ Community
- Android App Design Community

Here are the top two resources that you might find most useful:

- Lately, using the material design spec a lot for user interface development.
- Stack Overflow is definitely used the most for day-to-day development. I usually does a search in Google and click on any Stack Overflow results first.

Practice Set 2





Part 2: Using Variables

Goal: This program calculates the amount of sleep debt* in a week. The user estimates how much they sleep on a weekday/weekend day. The equation assumes you need 8 hours of sleep a night. The user in this case sleeps 5 hours on a weekday and 9 hours on a weekend.

```
int weekday = 5;
int weekend = 9;
int optimalHours = 7*8;
int actualHours = weekday;
actualHours = actualHours + weekend*2;
int solution = optimalHours - actualHours;
display(solution);
```

*Look at the Instructor Notes if you don't know this term.

Was the goal accomplished? Y N

What is displayed?

Part 2: Travel Times

Goal: This program is for commuters to better plan their morning schedule. This part of the program shows the average number of minutes it took to drive to work over the past 3 days.

```
int day1= 15;
int day2 = 22;
int day3 = 18;
display(day1+day2+day3/3);
```

Was the goal accomplished? Y N

What is displayed?

Scope of Variables

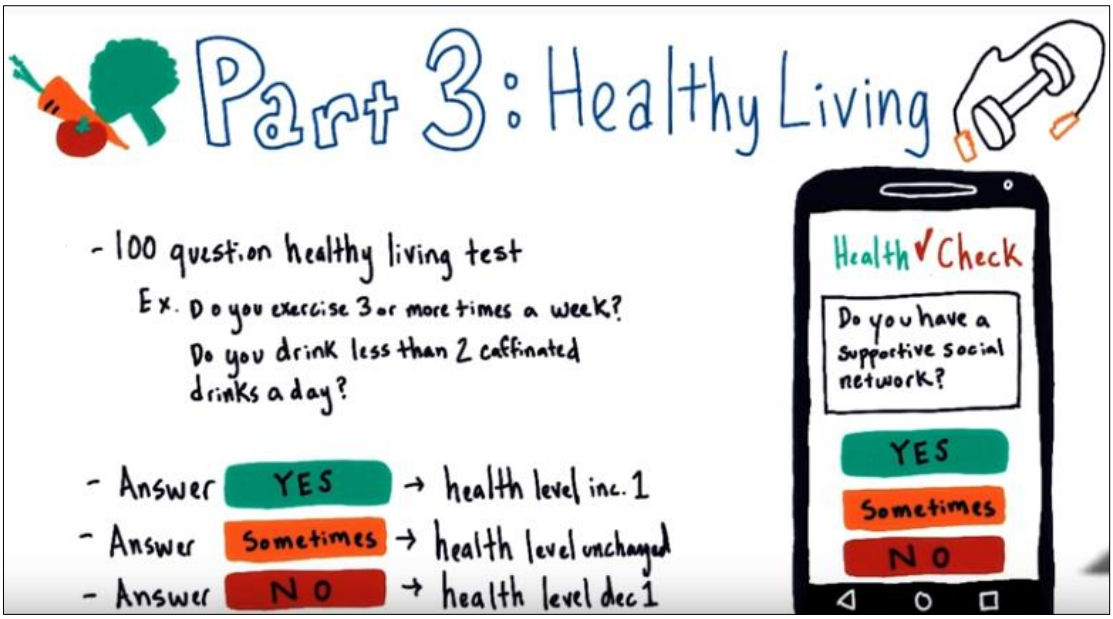
Part 3: Cola Wars


```
public class MainActivity extends AppCompatActivity {
....
public void voteUdacicola(View view)  {
 int udacicolaVotes = 0;
 udacicolaVotes = udacicolaVotes +1;
}
public void votePepcity(View view) {
 int pepcityVotes;
 pepcityVotes = pepcityVotes +1;
}
public void showMeVotes(View view) {
 display(udacicolaVotes + " vs. " + pepcityVotes);
}
...
}
```

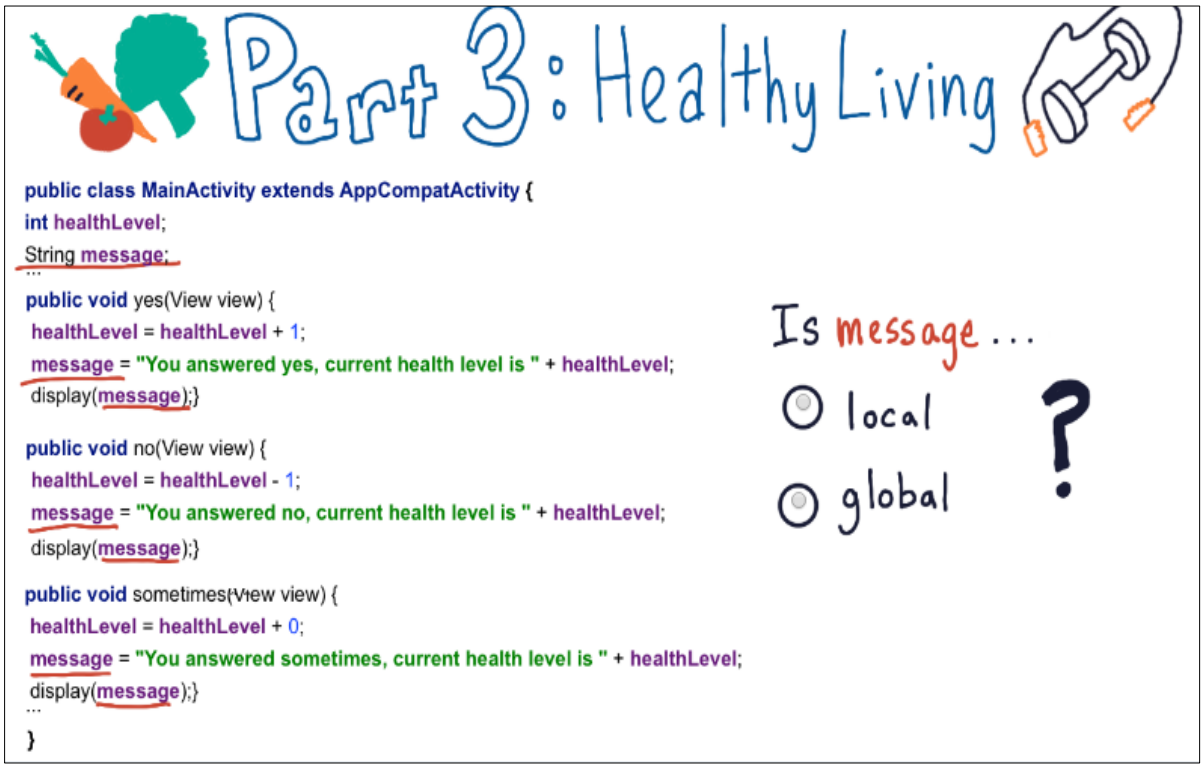
Is udacicolaVotes . . .

- local
- global

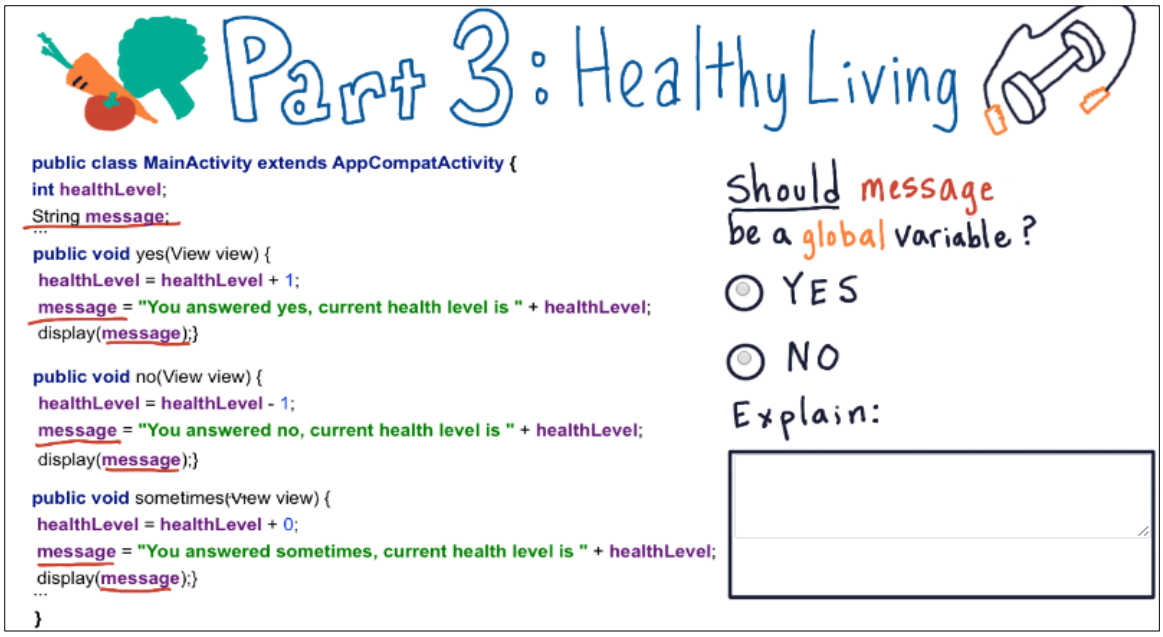
?



Look at this App above and solve this quiz, and not that when the user click yes or no a message will show what the user has chosen and what is the current health level.



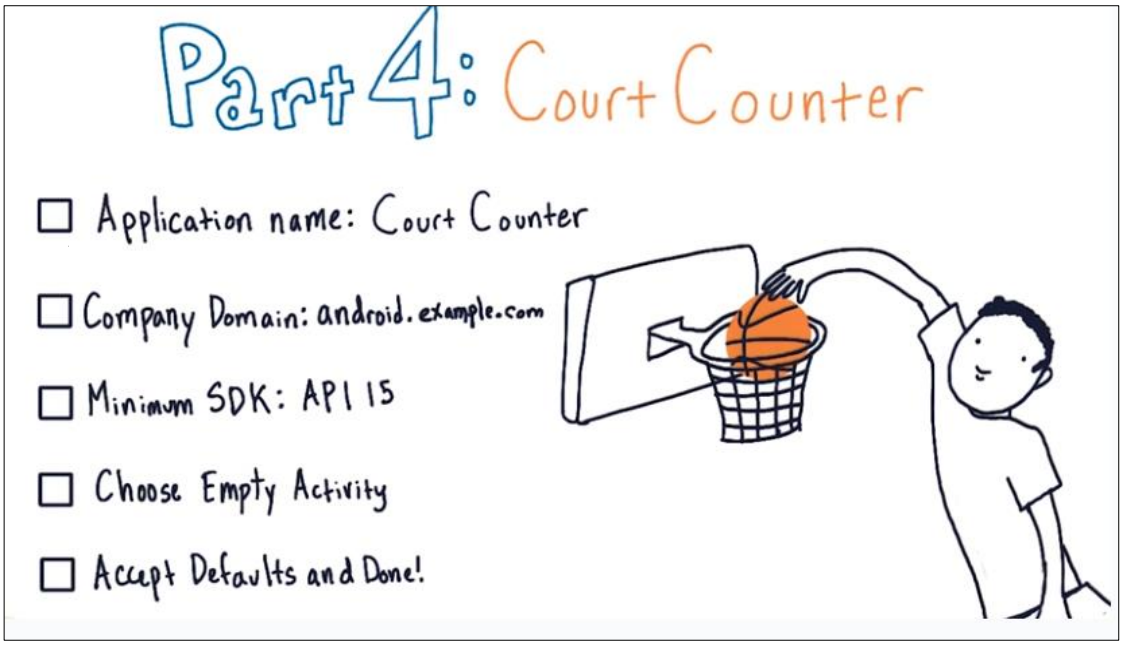
To get more understanding, try solving next quiz and think why you made the choice and why it is better than the other for this App.



Court Counter Intro

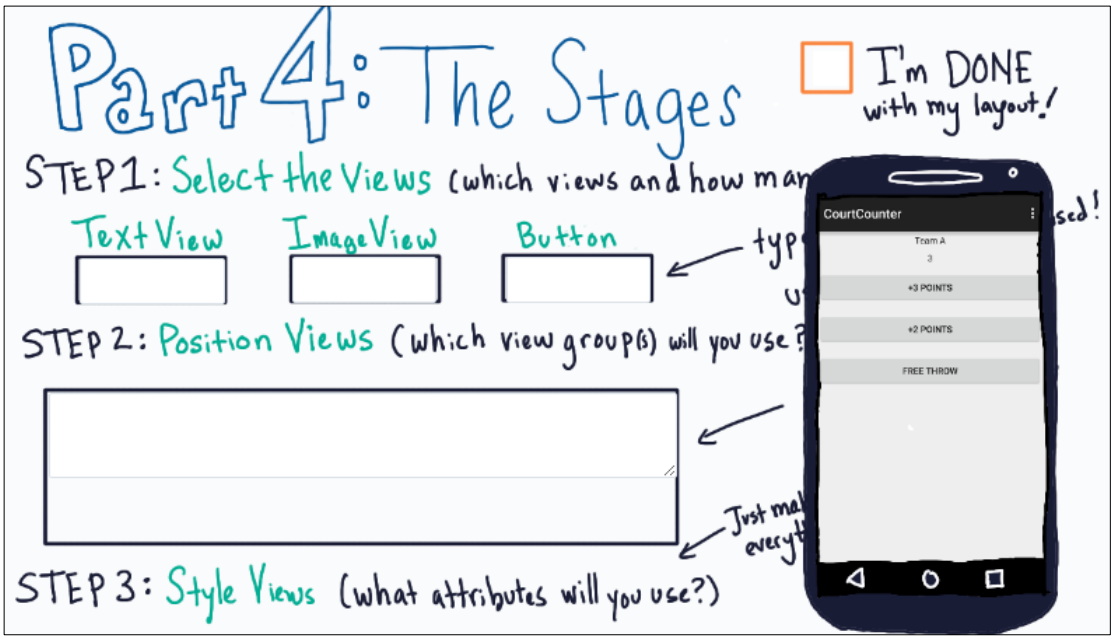
For the rest of this practice set you will work on the basketball app. The basketball scoring app called **court counter.** The purpose of this App is to keep track of the score of basketball game. The end App will look like this.

To start building this app I want you to go and do the followings steps that you have done many times before this time.



The Stages

Here are the steps that will help you to start creating the layout.



The two helpful attributes are **layout_gravity** and **gravity**.

Their names should help you to know the difference:

- `android:gravity` sets the gravity of the content of the `View` its used on.
- `android:layout_gravity` sets the gravity of the `View` or `Layout` in its parent.

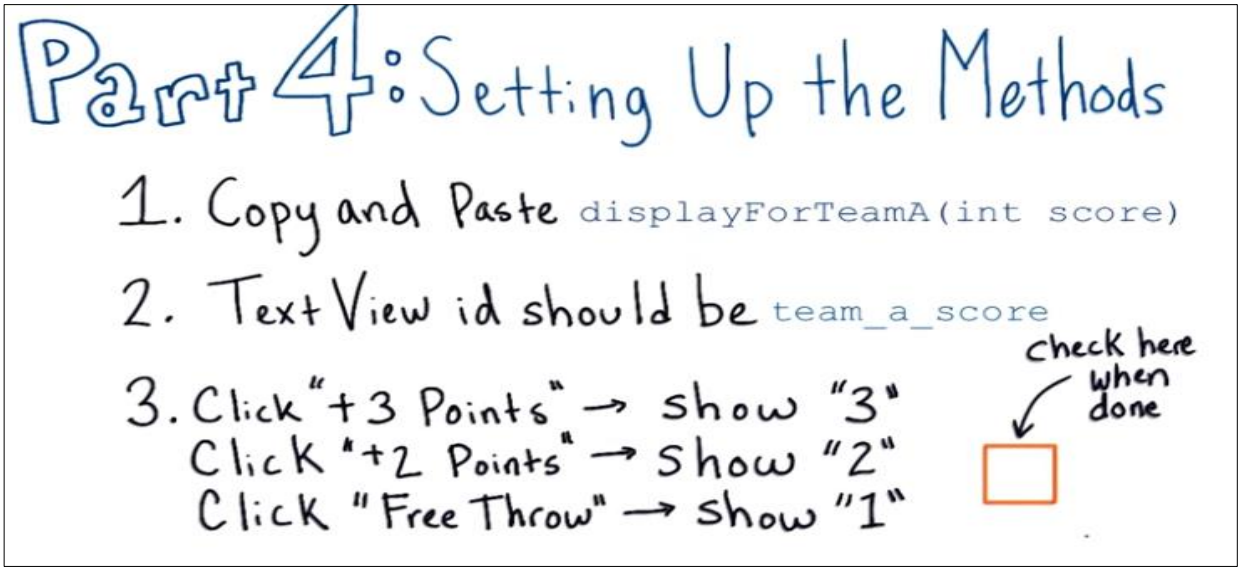
The XML code after this quiz should look like this.

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="4dp"
        android:text="Team A" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="4dp"
        android:text="0" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="+3 Points" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="+2 Points" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="Free throw" />
</LinearLayout>
```

Setting up the Methods

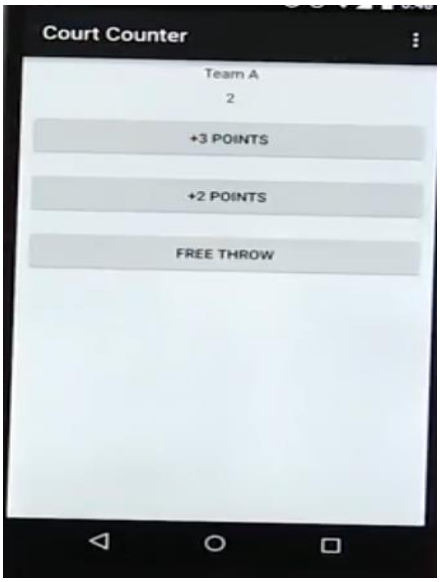
Now your layout appears good, but when pressing on these buttons nothing happens! It's your part now to do these tasks.



MainActivity.java

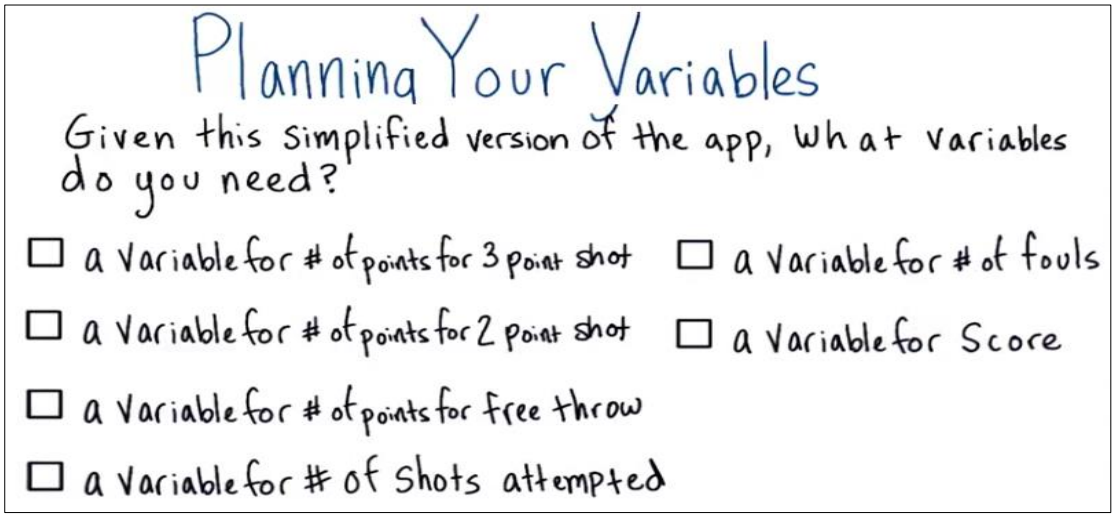
```
/**
 * Displays the given score for Team A.
 */
public void displayForTeamA(int score) {
  TextView scoreView = (TextView) findViewById(R.id.team_a_score);
  scoreView.setText(String.valueOf(score));
}
```

Planning Your Variables



Now this App is interactive but to not too much, it looks that we need a variable and I need you to think what variables that you will need. As you can see this version of App we have only team A, and we don't have ability to Reset the score. At this moment, what variables/variable we should use to make the number increased when we click on buttons?

Here is a quiz to check yourself for this task.



Here "#" = "number"

In programming languages a **constant** is a value that is never changed or modified during the course of the program. In Java you can force a value not to be changed by using the keyword `final`. For example:

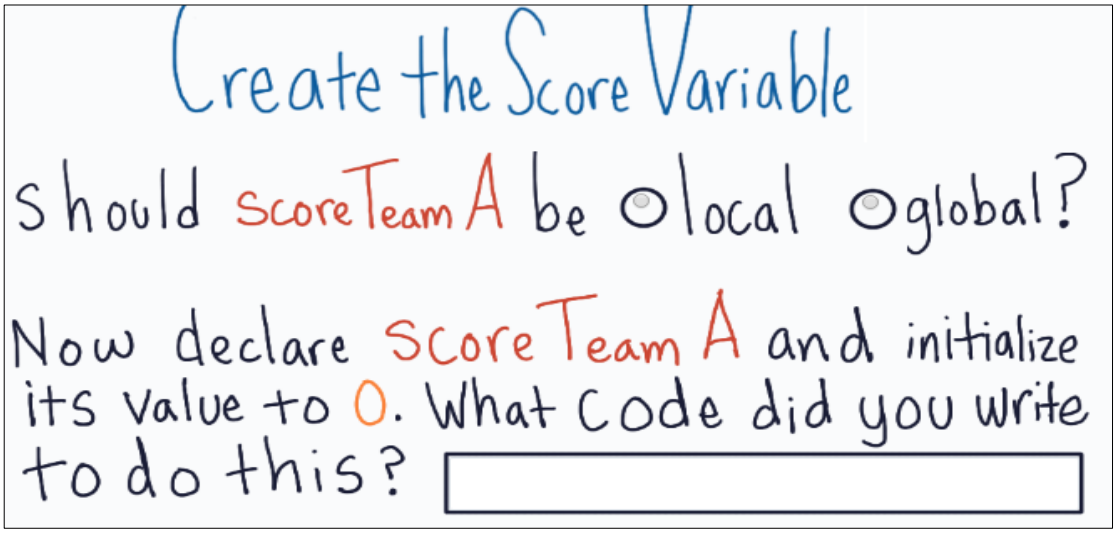
```
final int POINTS_FOR_FREE_THROW = 1;
```

Then if you wrote the following, you'd get an error:

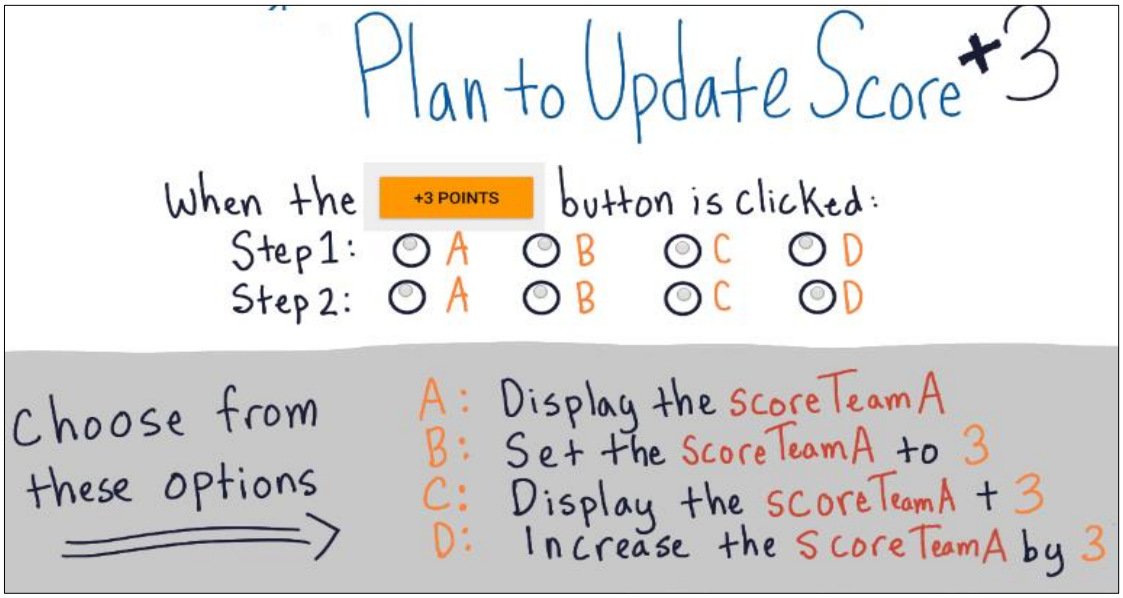
```
POINTS_FOR_FREE_THROW = 100;
```

Note, by convention, constants have all capitalized letters in their names. Also, instead of spaces or camel-case, underscores are used between words. You can declare variables as constants to keep yourself from accidentally changing the value of a variable. For a more detailed explanation of final variables,

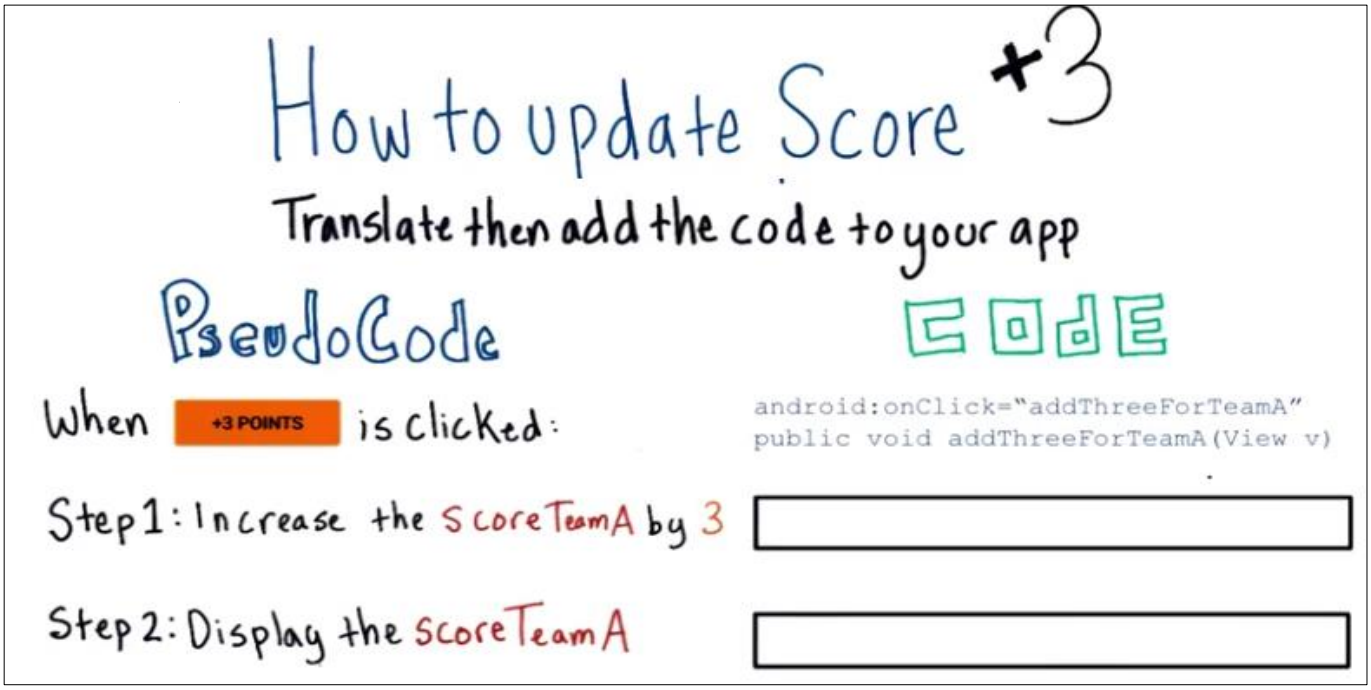
You want a variable to track the score and that's the only variable that you'll need for now. Now you might be wondering why I don't call this variable "score" that's because we will have later other team with different score. So we will be explicit and call it "**scoreTeamA**". In any game the score start with "**0**" so I want you to initialize the variable with "0". For now do this **quiz** to make sure that you can proceed.



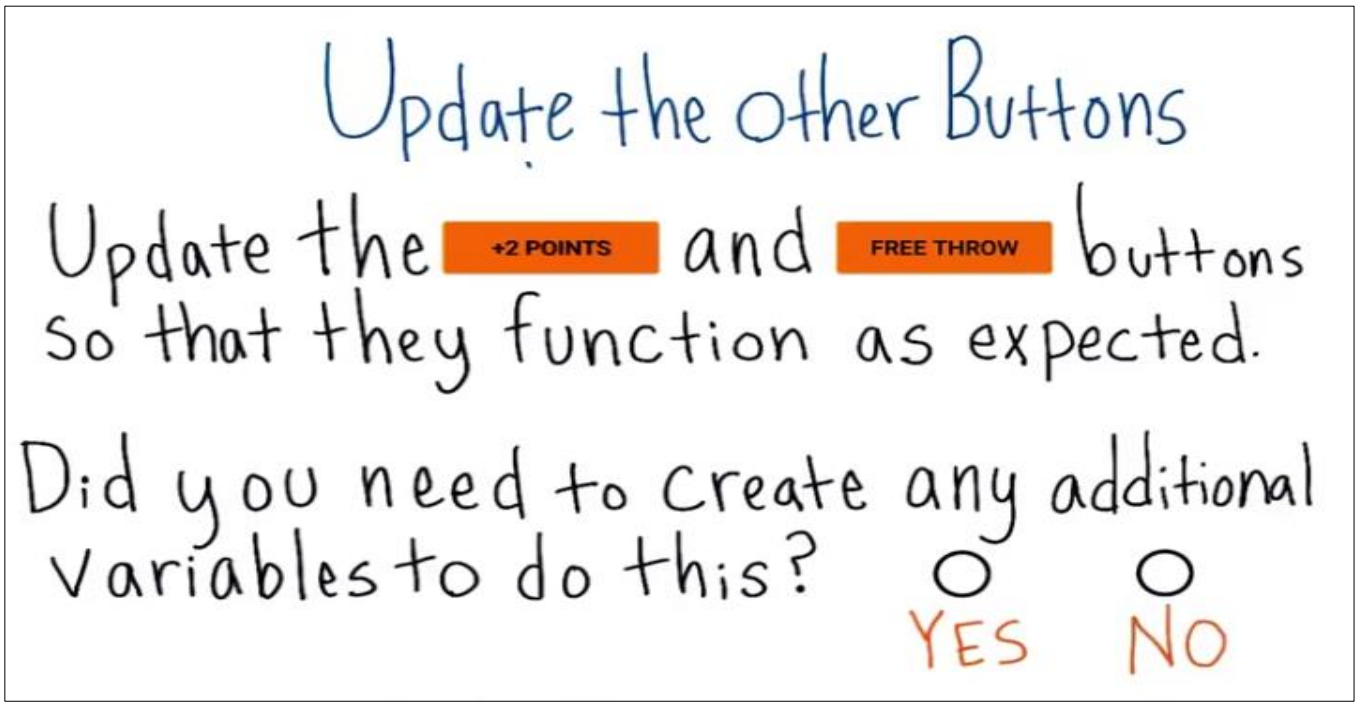
Until this moment nothing happens when you click on buttons, I want you to tell me how you can make proper planning (Pseudocode) on how "**+3 button**" should work? Do this quiz to continue …



Now you should translate Pseudocode to java code and add it to your App, in this quiz you have one Pseudocode and its translation to java code, try to do the other two steps.



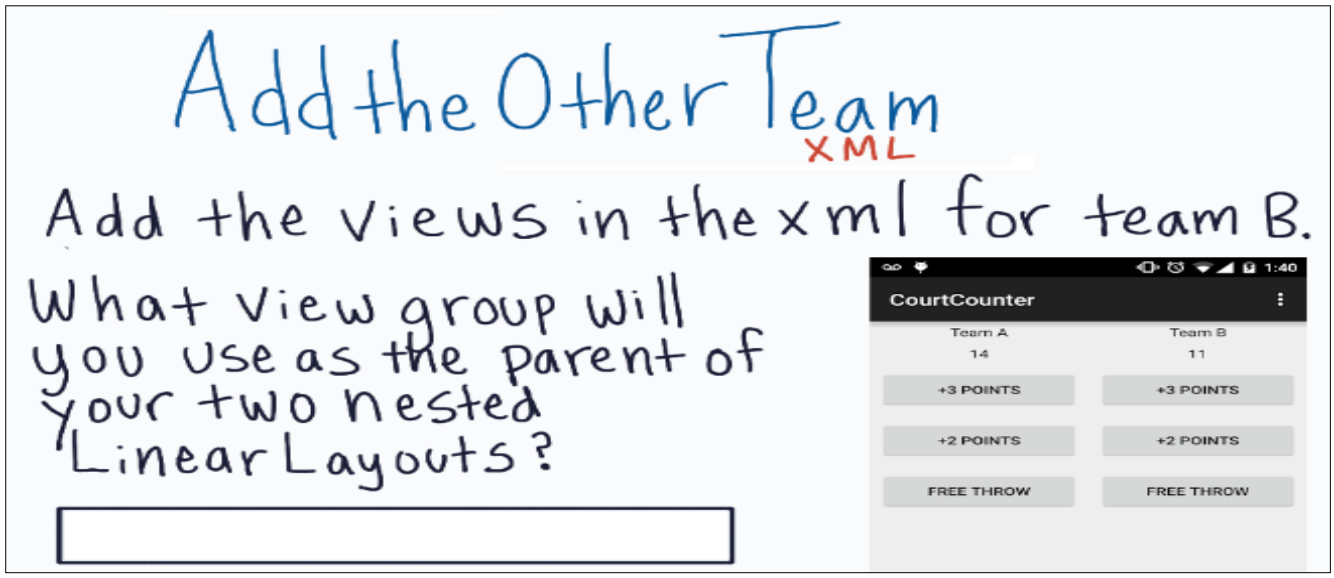
Alright I expect you have changed to code so the +3 button would actually add 3 points and continually increase. Now do the other part by this quiz.



You should not need to add any additional variables. The +**2 Points** and **Free Throw** buttons will need to change the **scoreTeamA** variable only.

Ok good work, unhappily your App is works for one team for now?

What we want is to make it works for two teams A and B. So let's start with the **XML** and later we can work on **Java**. Do this quiz to understand how you can make the new design by using nested layout.



At this time our XML is neat, you are going to do the right side update for Team B, do these steps:

Quiz: Add The Other Team - Java

Step 1 : Add the Display Code

Add the following method inside of `MainActivity.java`

```
/**
* Displays the given score for Team B.
*/
public void displayForTeamB(int score) {
 TextView scoreView = (TextView) findViewById(R.id.team_b_score);
 scoreView.setText(String.valueOf(score));
}
```

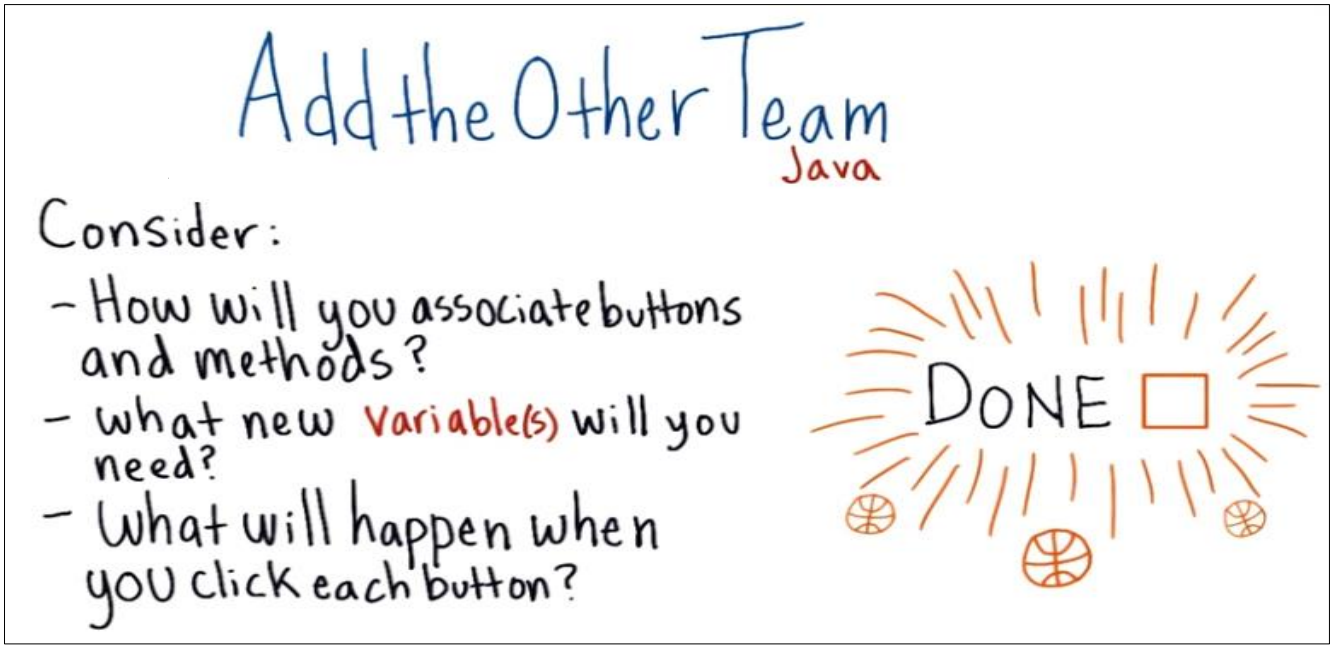
Step 2 : Set the Correct ID

Set Team B's score TextView to the id `team_b_score` using the id attribute. This is the code you will use:

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

Now you can use the code `displayForTeamB()` to show Team B's score in the TextView with the id `team_b_score`. For example `displayForTeamB(12)` will display a score of 12.

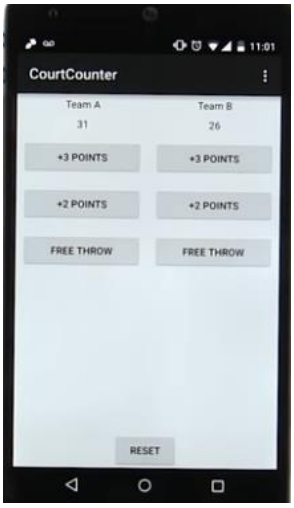
Add the Team B to Java, by taking these steps into consideration.

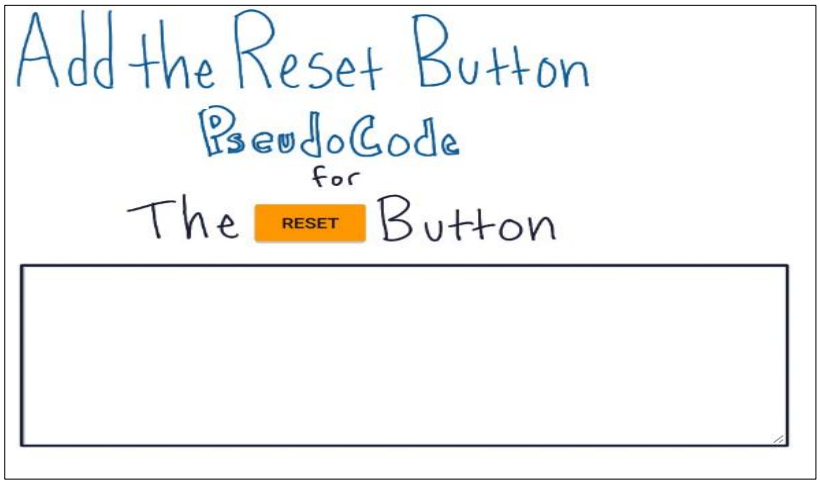


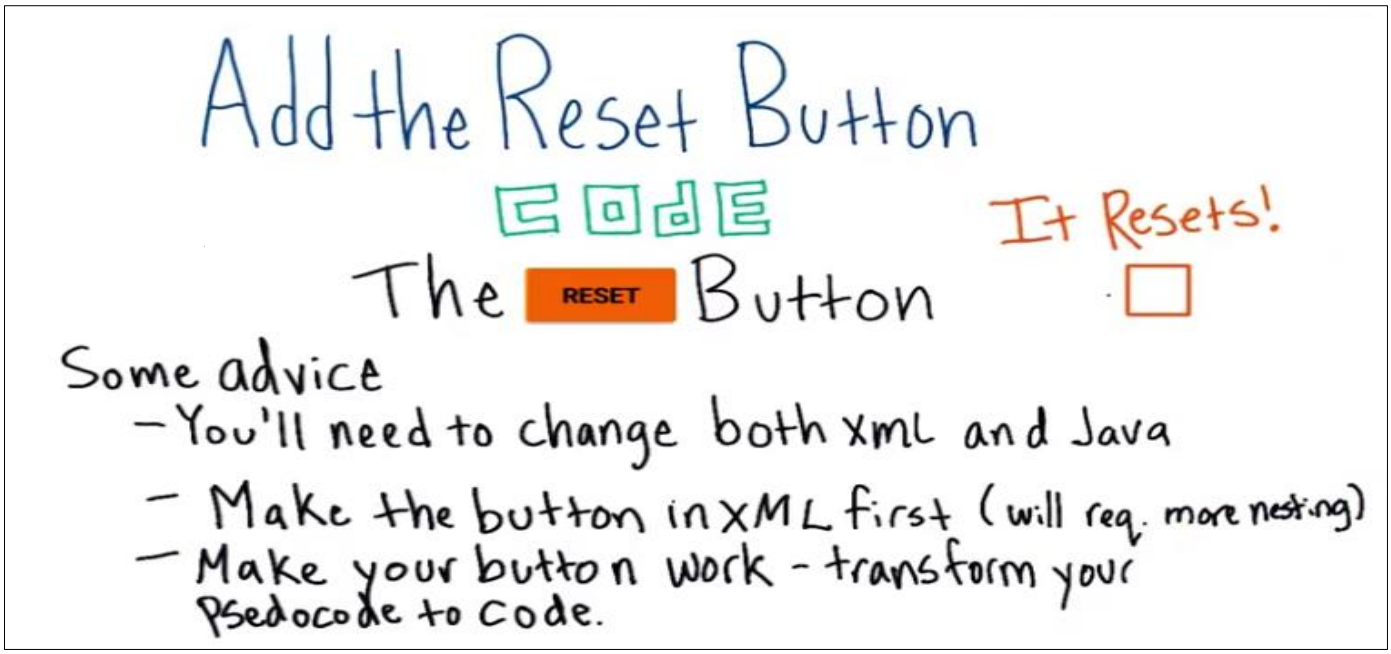
Plan to Add the Reset Button

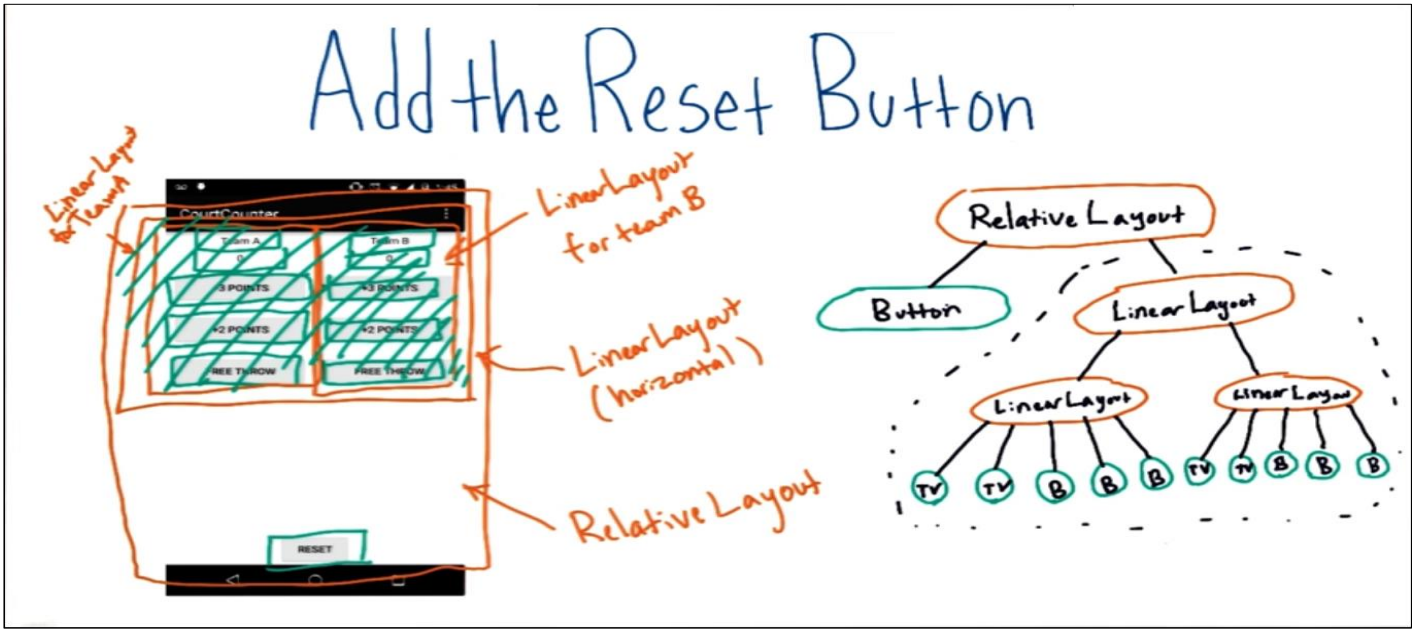
Till now you are doing well and the functionality of the App is almost finished. We still have one more thing to add before styling the App, and that's adding the **Reset button**.

The functionality of **Reset button** is when the match is end, and a new match will start you hit the **Reset button** and the two scores of the teams set to **Zero**. Before you start coding I want you to do your Pseudocode, and I will not help you in doing this to make it real test for you. Write your Pseudocode inside the empty box.









Try add this code to your Java to Make reset button work and don't forget to write XML code: android: onClick="resetScore"

```
/**
 * Resets the score for both teams back to 0.
 */
public void resetScore(View v) {
    displayForTeamA(scoreTeamA);
    displayForTeamB(scoreTeamB);
    scoreTeamA = 0;
    scoreTeamB = 0;
}
```

Let's say team A has score 30 and Team B 40 , when I say display for Team A it's going to show 30 and when I say display for tem B it's going to show 40 only after that it actually sets the values to zero. But since it displayed first it's not going to show the values of zero but the variable will be zero so when I press something like add 3 for team B it's going to take the value of zero that I set down here and then display three. So to us it looks like Team B went from having 30 then to after clicking the reset button, a score of three which is a kind of strange jump. Now one way to fix this by writing this code:

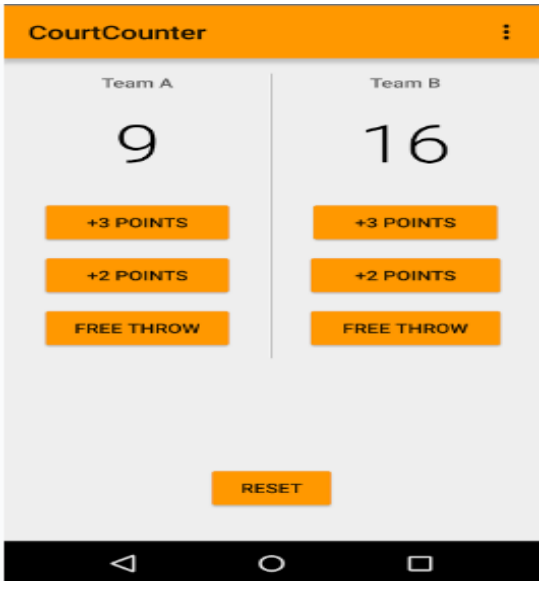
```
/**
 * Resets the score for both teams back to 0.
 */
public void resetScore(View v) {
    displayForTeamA(0);
    displayForTeamB(0);
    scoreTeamA = 0;
    scoreTeamB = 0;
}
```

And for better coding you can revert the code to be like this:

```
public void resetScore(View v) {
    scoreTeamA = 0;
    scoreTeamB = 0;
    displayForTeamA(scoreTeamA);
    displayForTeamB(scoreTeamB);
}
```

Make your App Pretty

For this quiz you'll be making your app look like this:

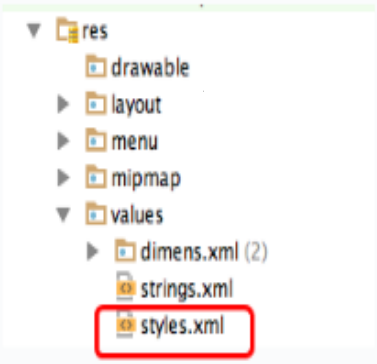


We'll do this in five steps:

1. Copy over new contents for the styles.xml file.
2. Make a grey line between the two LinearLayouts.
3. Adjust the LinearLayouts sizes.
4. Update the text size, color and font.
5. Add the correct padding and margin.

Step 1 : Copy Over New styles.xml File

The `styles.xml` file is a type of resource file that defines the format and look for a layout. You can set things like button colors and the color of the action bar.



The XML code to copy is below.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light">
        <!-- Primary theme color of the app (sets background color of app bar) -->
        <item name="colorPrimary">#FF9800</item>
        <!-- Background color of buttons in the app -->
        <item name="colorButtonNormal">#FF9800</item>
    </style>
</resources>
```

`colorPrimary` will change the color of the Action Bar. `colorButtonNormal` will change the color of the Buttons.

Note: colorButtonNormal will only work on phones running API 22 and above. So if you have an older phone, the buttons will not be colored orange.

Step 2 : Make the Grey Line

You can use the `View` tag to make a view box and you can then color in the box using the `background` attribute. The color of the view should be `@android:color/darker_gray`. By making the box 1dp wide, it will look like a line.

Figuring out where exactly to position the box is up to you.

Step 3 : Adjust the LinearLayouts

When you first add the view, you might see that the layout looks like this:

You'll need to adjust the sizes of various views so that the grey line extends only to the end of the buttons (don't worry about the top of the line, you'll fix that in step 5). **DO NOT** use fixed widths, instead use `match_parent` and `wrap_content` only.

Step 4 : Update the Text Size, Color and Font

Update the the TextViews to the following specifications.

Team Name TextViews:

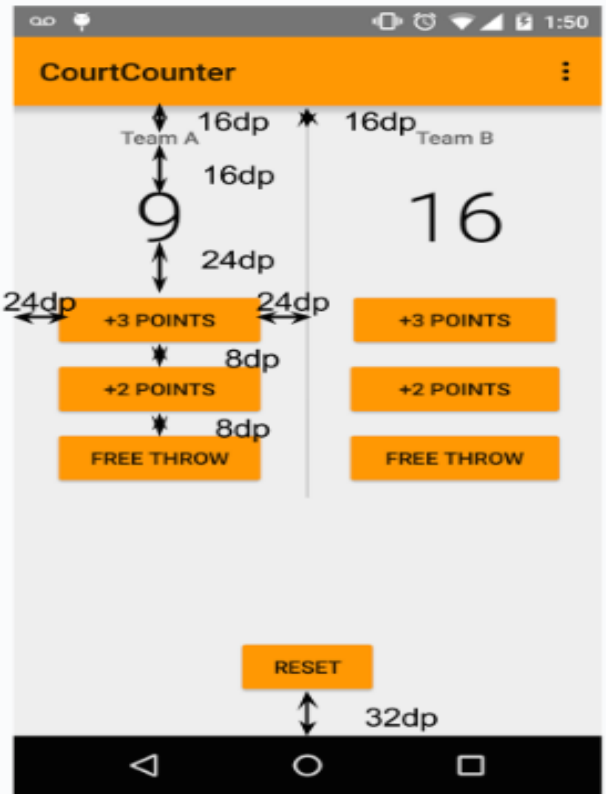
- **Size** 14sp
- **Color** #616161
- **Font** sans-serif-medium

Score TextViews

- **Size** 56sp
- **Color** #000000
- **Font** sans-serif-light

Step 5 : Padding and Margin

Update the layout to have the correct spacing, as shown in the diagram:



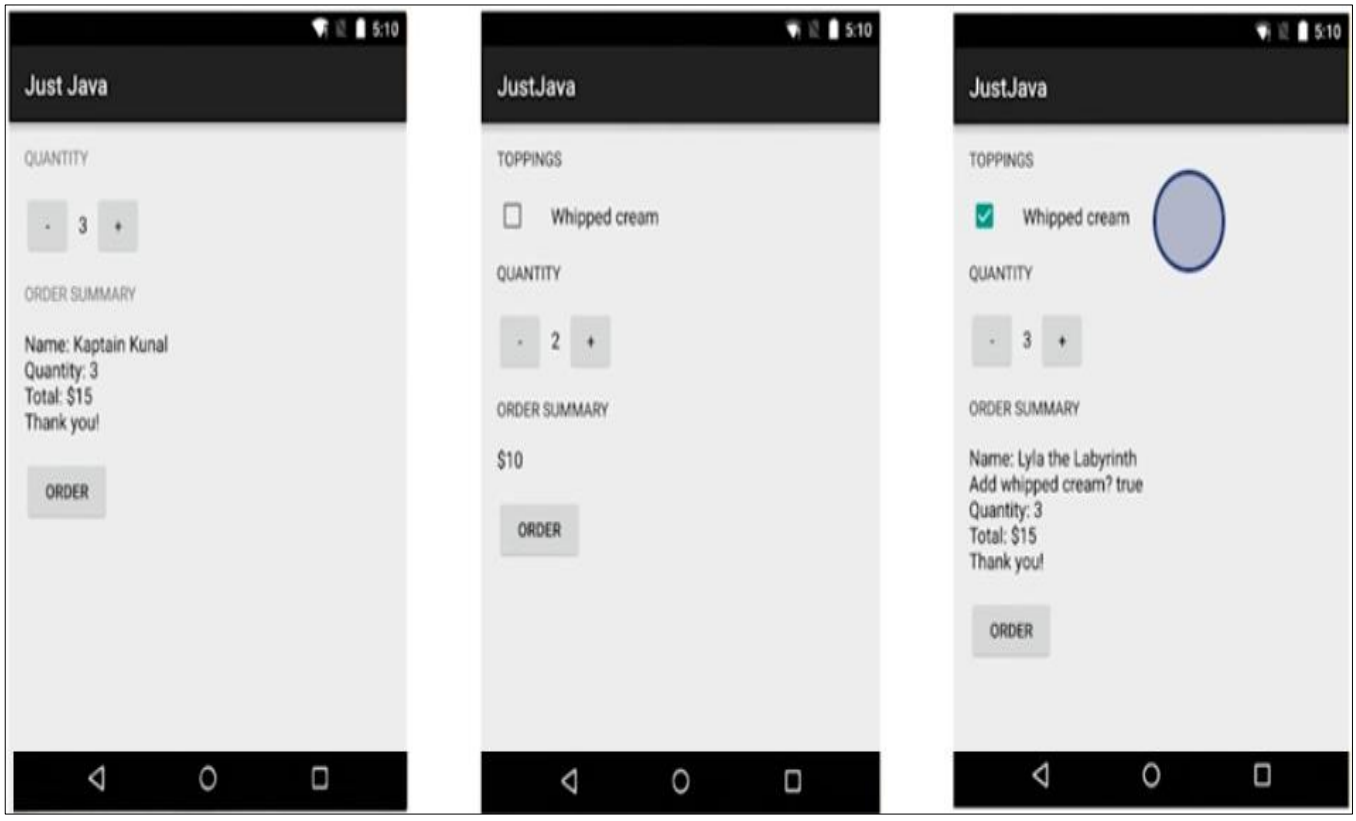
Recap

Congratulations! You're done with Practice Set 2! As a quick recap, in this practice set you:

1. Reviewed the rules of variable initialization and declaration
2. Interpreted program output using hand simulation
3. Determined the appropriate variable scope for a variable
4. Made a sweet basketball score counting app.

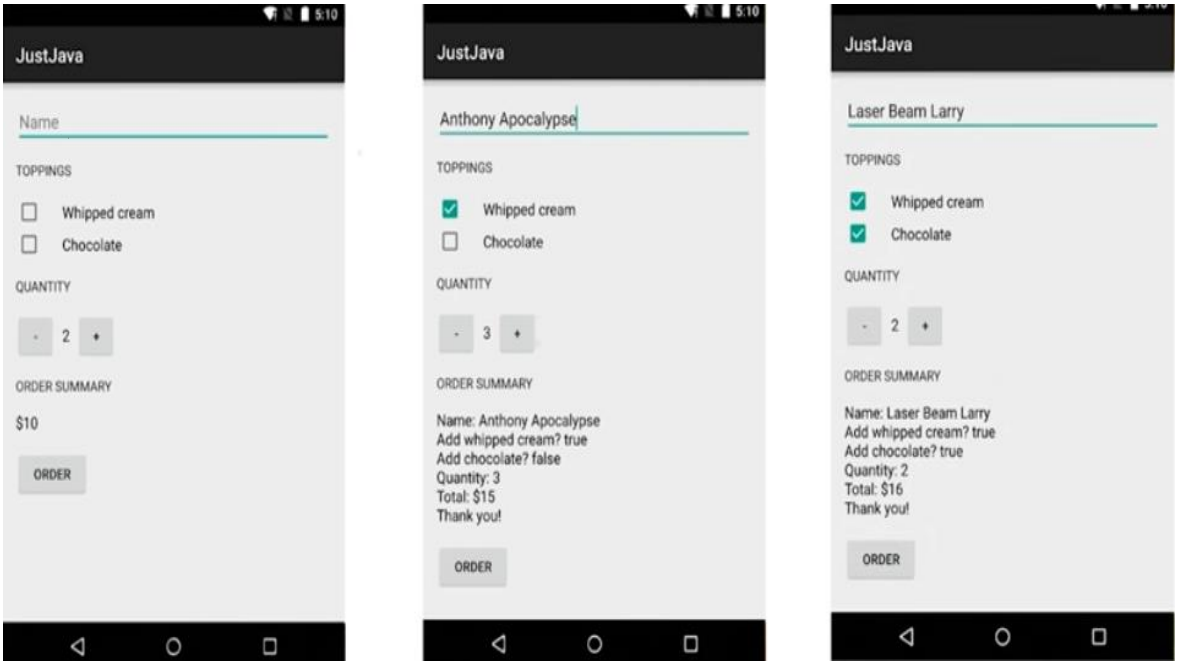
Quiz App

The goal of this part is to design and implement a short quiz app about a topic that you are familiar with. As you remember we done with happy New Year App, then we started with the coffee ordering Ap. For this part we will start with **Methods**. Below are a look on different stages of the App for this part. We will start by adding an order summary section to the App and understand how methods in Java can help us write more efficient code. Later we'll add a **checkbox** for a topping like whipped cream to add to our coffee.
However we need to understand **object-oriented programming** to determine the state of the checkbox whether the user checked it or not, and include that information in the order summary. Within our app, **object-oriented programming** allow us to dynamically interact with Views , so that we can read the state of the views then change the content of other views while the app is running.

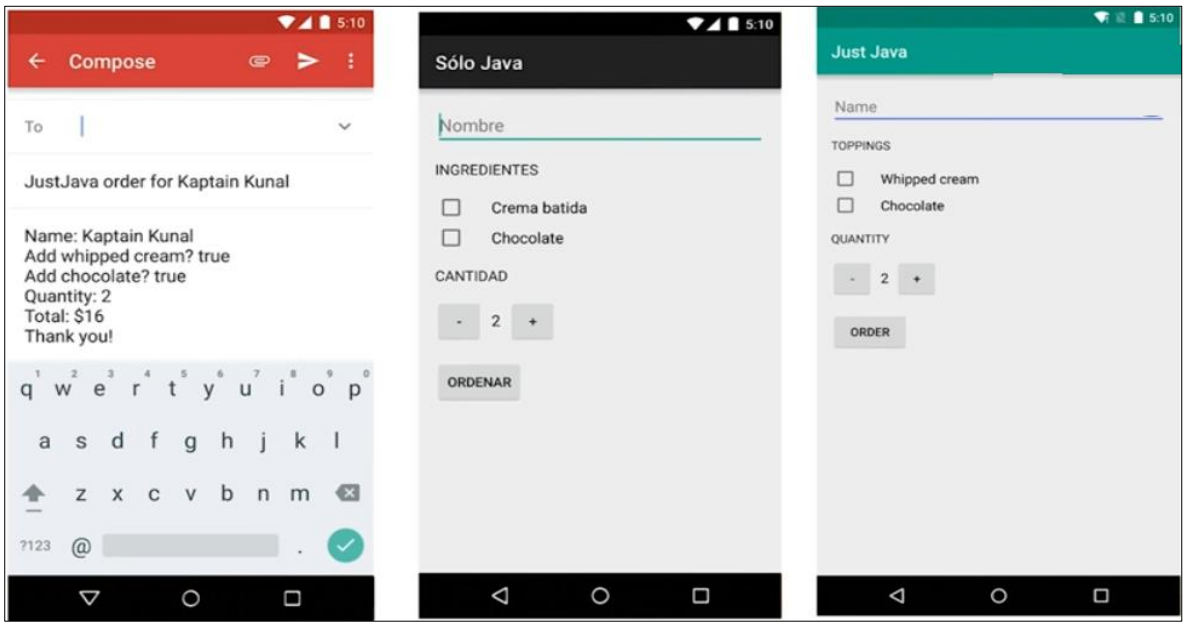


As you can see in the figure next page, we'll add another **checkbox** for the chocolate topping and then we'll add a **text field,** so that we can ask users for their names. Again we'll write Java code to read in the user input from this field and then display it in **the order summary.** We'll also learn about control flow statements particularly something called if/else statements, so that our App has different behavior based on certain conditions. Basically by looking at what

toppings the user wants. We'll be able to adjust the price per cup of coffee either by $1 or $2.



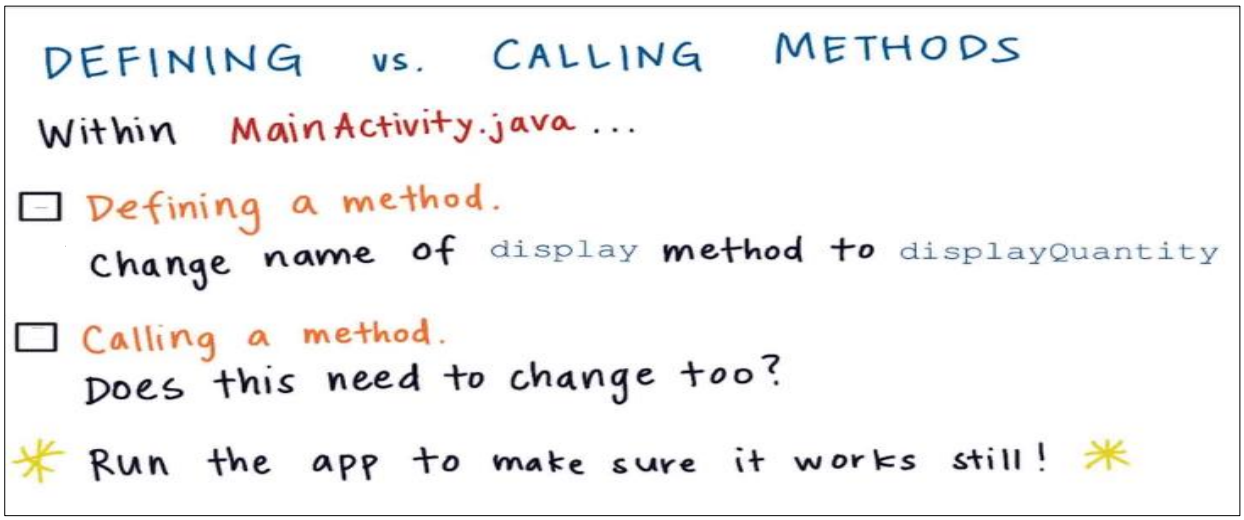
Now the order summary is not that useful if it never leaves the app. So we'll learn about **intents** in Android, so that when user hits the Order button, we can send the order summary out **to an email app** like this first one screen below. Then you can specify a specific email address to send it to. **We'll end this part/lesson** by covering more topics on **Android resources**, learning about how to support different languages in the app such as Arabic. We'll also experiment with changing the styles and theme of the App. For example you can change the **color of the App bar** by customizing the **theme** of the App. There's a lot to learn so let us start!



Probably you know driving a car but at the same you don't know how it works exactly on details, because you only need to put the key and drive it. This process is similar to writing code, when you use a method to do something. The main idea is that you can define a method somewhere and that's like building the car and later you can call that method and that's like driving the car.
The difference between writing a method and using its code only without defining a method is that you will need to duplicate the code many times instead of just calling the method if it is available. For example the display method that we have used in the cafeteria ordering app, if we didn't write that method we will need to duplicate its code every time we need to display the quantity value. It's important to know how to go to method declaration in Android studio by right click on a method and look this menu in the diagram.

This scrolls the file to where the method is defined, and this is useful because you can see the actual instructions that are contained within that method. For other methods it might open a declaration in the Android framework code. This was written by the Android team the TextView.Java file.

To help you understand the difference between defining and calling a method, let's do this quiz in next page.

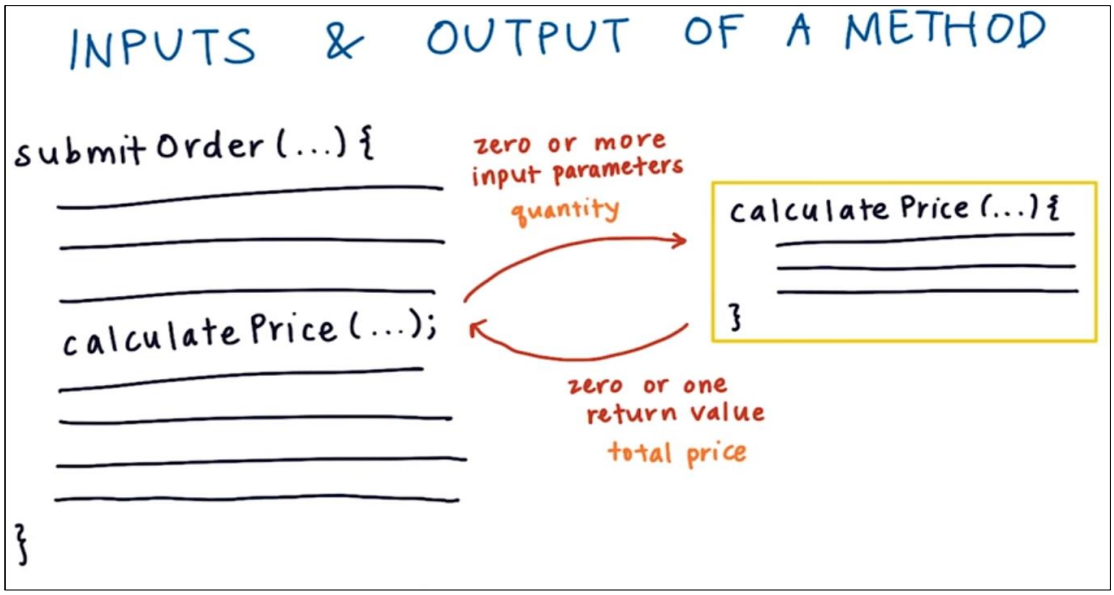


Defining Vs Calling A Method
The keyboard shortcut for going to a method declaration in Android Studio is :

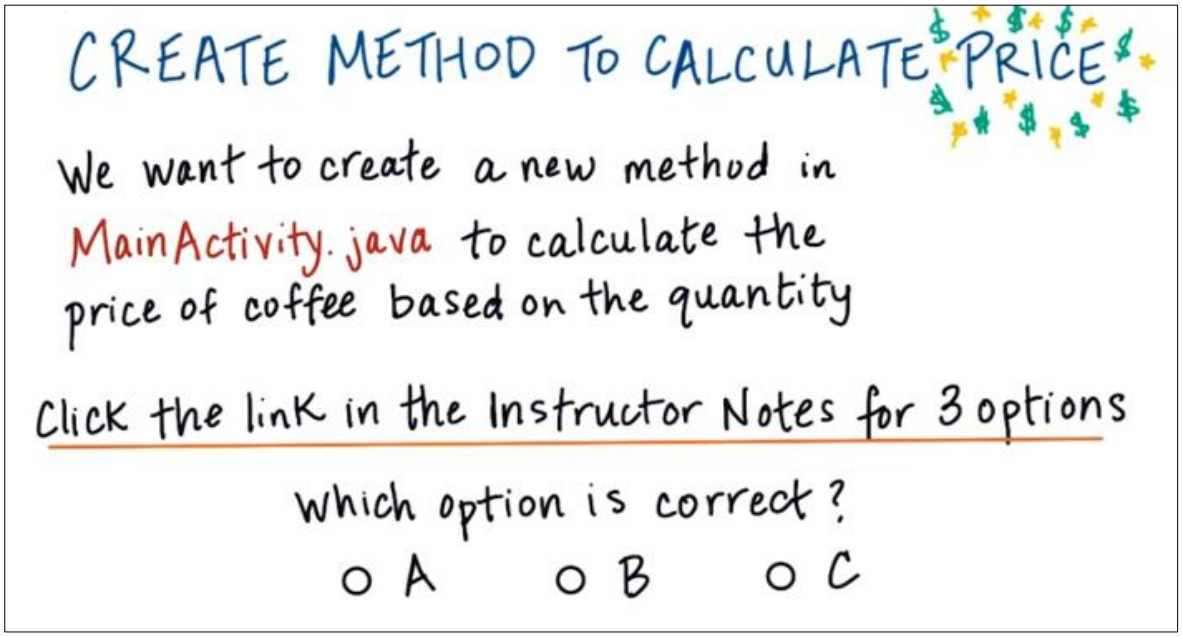
- On Mac **command+b**
- On Windows **control + b**

Calculate Price Method

Let's look at the submitOrder method in the Main Activity as example, so you have a bunch of code in that method and when the user click on Order button it will cause the submitOrder method to be called then Android studio will execute each instruction from top to bottom. When calculatePrice method is called it will jump over to where calculatePrice method is defined then instructions inside this method are executed until last line is finished then Android Studio moves back to the instructions after the calculatePrice method.



This is the first time that we create a new method on Main Activity on our own. Instead of starting from scratch we want to start learning how to recognize the correct method by reading some code snippets. Read the 3 options then determine which option is correctly implement the method that calculates the price of the order.



With the time and when you get more experience you will know when you need to implement new methods and for what you should do.

Defining Methods (https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html)

Here is an example of a typical method declaration:

```
public double calculateAnswer(double wingSpan, int numberOfEngines,
                              double length, double grossTons) {
    //do the calculation here
}
```

The only required elements of a method declaration are the method's return type, name, a pair of parentheses, `()`, and a body between braces, `{}`.

More generally, method declarations have six components, in order:

1. Modifiers—such as `public`, `private`, and others you will learn about later.
2. The return type—the data type of the value returned by the method, or `void` if the method does not return a value.
3. The method name—the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, `()`. If there are no parameters, you must use empty parentheses.
5. The method body, enclosed between braces—the method's code, including the declaration of local variables, goes here.

Naming a Method

Although a method name can be any legal identifier, code conventions restrict method names. By convention, method names should be a verb in lowercase or a multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, etc. In multi-word names, the first letter of each of the second and following words should be capitalized. Here are some examples:

```
run
runFast
getBackground
getFinalData
compareTo
setX
isEmpty
```

Typically, a method has a unique name within its class. However, a method might have the same name as other methods due to *method overloading*

Option A

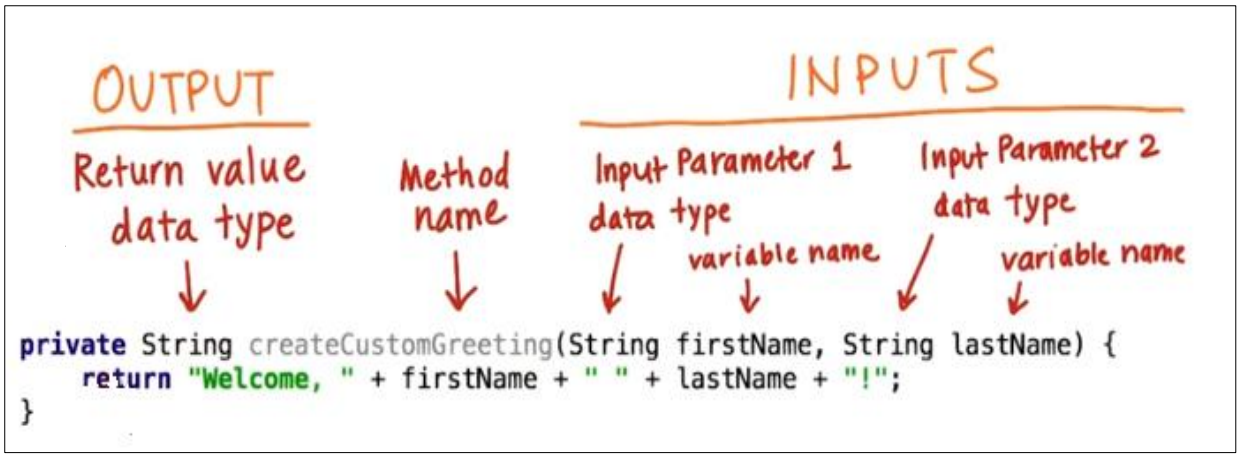
```
/**
 * Calculates the price of the order based on the current quantity.
 *
 * @return the price
 */
private int calculate price(int quantity {
    int price = quantity * 5;
    return price;
}
```

Option B

```
/**
 * Calculates the price of the order based on the current quantity.
 *
 * @return the price
 */
private calculatePrice(int quantity)
    int price = quantity * 5;
    return price;
```

Option C

```
/**
 * Calculates the price of the order based on the current quantity.
 *
 * @return the price
 */
private int calculatePrice(int quantity) {
    int price = quantity * 5;
    return price;
}
```



Now after you have looked at the previous example, go and make the other methods for this table.

DEFINE A METHOD

Fill in blanks. Refer to code in instructor notes.

	Method Name	Input Parameters	Return data type
Sample	createCustomGreeting	String firstName, String lastName	String
Method 1			
Method 2			
Method 3			

Method 1

```
private String createCalendarEventReminder(String eventName, String location, int minutesAway) {
    String reminder = "You have an upcoming event in " + minutesAway + " minutes.";
    reminder = reminder + " It is " + eventName + " held at " + location + ".";
    return reminder;
}
```

Method 2

```
private int deductPoints(int pointsUsed) {
     // Everyone starts with 100 points
     int numberOfPoints = 100 - pointsUsed;
     return numberOfPoints;
}
```

Method 3

```
private String findTotalTripLength(int distanceOfFirstTrip, int distanceOfSecondTrip, int distanceOfThirdTrip) {
     // Assume we need 2 miles to go to our friend's home (where the trip will start).
     int totalLength = 2;

     // Then start adding in the trip length.
     totalLength = totalLength + distanceOfFirstTrip + distanceOfSecondTrip + distanceOfThirdTrip;

     // Form a string to display the total trip length.
     String message = "The total trip will be: " + totalLength + " miles.";
     return message;
}
```

Inputs to a Method

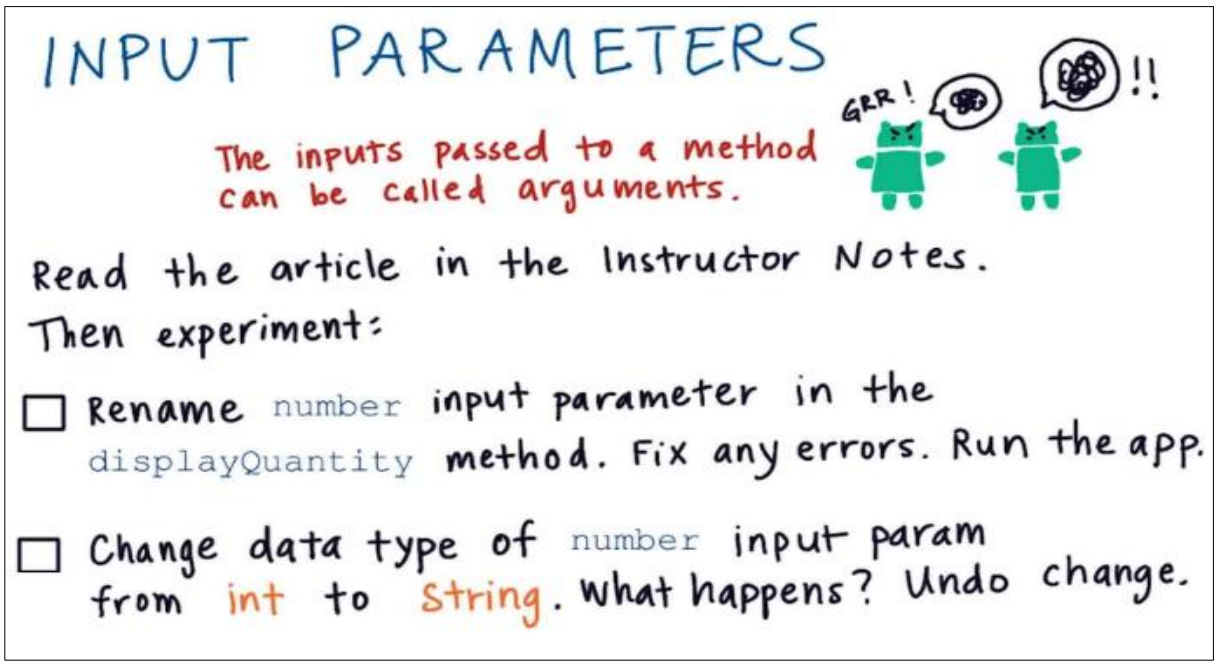
Passing Information to a Method or a Constructor

The declaration for a method or a constructor declares the number and the type of the arguments for that method or constructor. For example, the following is a method that computes the monthly payments for a home loan, based on the amount of the loan, the interest rate, the length of the loan (the number of periods), and the future value of the loan:

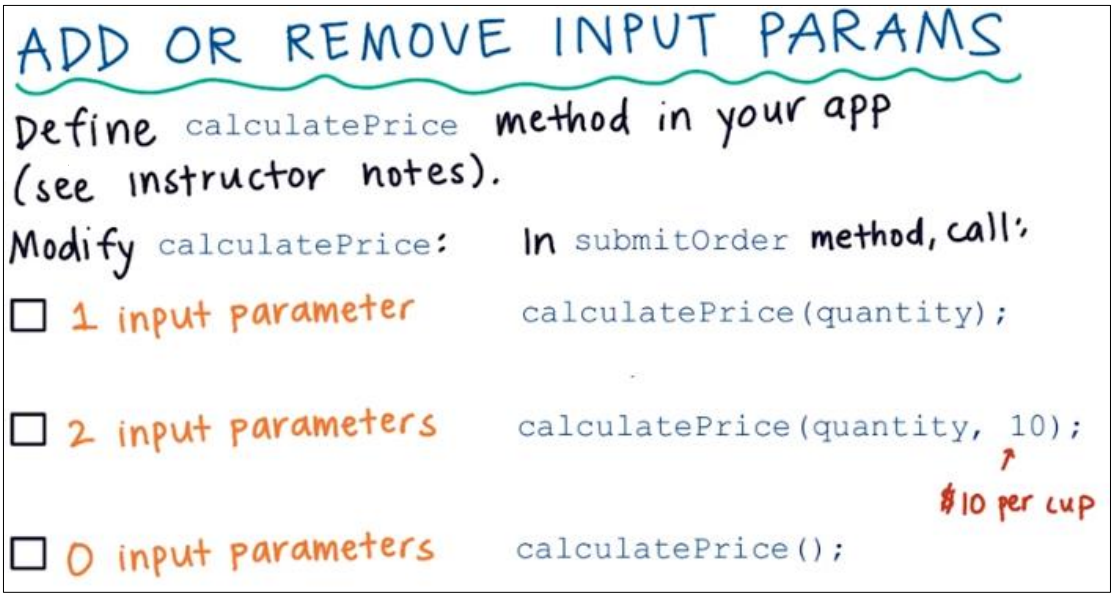
```
public double computePayment(
                    double loanAmt,
                    double rate,
                    double futureValue,
                    int numPeriods) {
    double interest = rate / 100.0;
    double partial1 = Math.pow((1 + interest),
                    - numPeriods);
    double denominator = (1 - partial1) / interest;
    double answer = (-loanAmt / denominator)
                    - ((futureValue * partial1) /
denominator);
    return answer;
}
```

This method has four parameters: the loan amount, the interest rate, the future value and the number of periods. The first three are double-precision floating point numbers, and the fourth is an integer. The parameters are used in the method body and at runtime will take on the values of the arguments that are passed in.

Note: *Parameters* refers to the list of variables in a method declaration. *Arguments* are the actual values that are passed in when the method is invoked. When you invoke a method, the arguments used must match the declaration's parameters in type and order.



Add or Remove Parameters



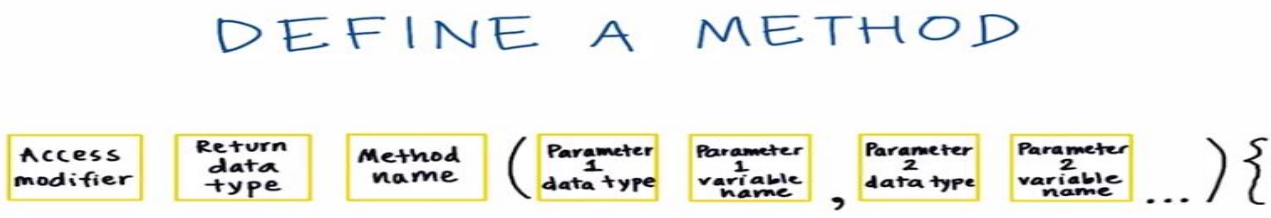
Let's do the no.1 by using the code below, copy this code below submitOrder method and you can put an empty line between each method to make it easy to read. The you can call this method inside submitOrder and pass **quantity** because we have one input parameter.

Calculate price method

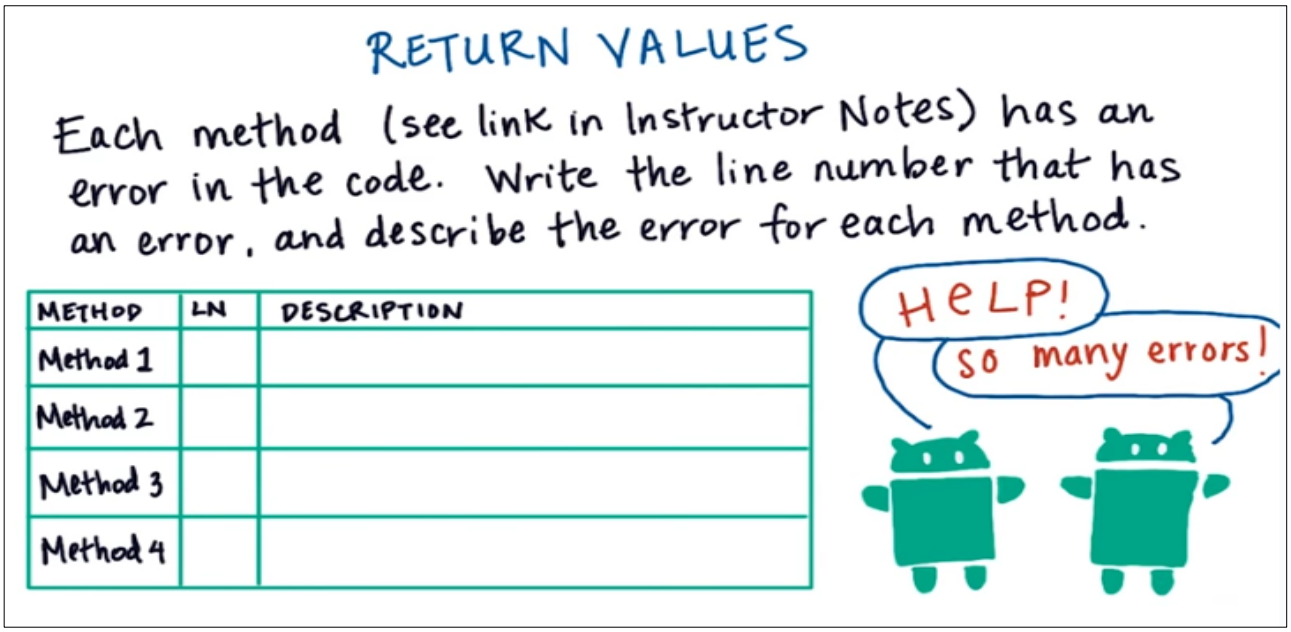
```
/**
 * Calculates the price of the order.
 *
 * @param quantity is the number of cups of coffee ordered
 */
private void calculatePrice(int quantity) {
    int price = quantity * 5;
}
```

Now do the same for **the 2 remaining task** by modifying input parameters.

Output from a Method




```
private int calculatePrice(int quantity) {
```



Method 1

```
/**
 * Get the email account name.
 *
 * @return the name of the account.
 */
private String getAccountName() {
    return "android@gmail.com";
    return "droid@gmail.com";
}
```

Method 2

```
/**
 * Add $4 of tip onto the current bill.
 *
 * @return the total price of the bill (including tip).
 */
private addTip(int bill) {
    return bill + 4;
}
```

Method 3

```
/**
 * Sets up the app for the current city.
 */
private nothing setup() {
    cityName = "London";
}
```

Method 4

```
/**
 * Get the number of students in a class.
 *
 * @return the number of students.
 */
private int getStudentClassSize() {
    return "20";
}
```

Returning a Value from a Method

A method returns to the code that invoked it when it

- completes all the statements in the method,
- reaches a `return` statement

For this coding task, you will making sure calculatePrice method is defined and called properly in the JustJave App, when you done check these boxes.

CALCULATE PRICE METHOD

Defining a method

- ☐ Modify `calculatePrice()` to have correct return data type. Assume 0 input parameters.
- ☐ Modify `calculatePrice()` to return correct price of order (quantity × price per cup). Assume $5 per cup.

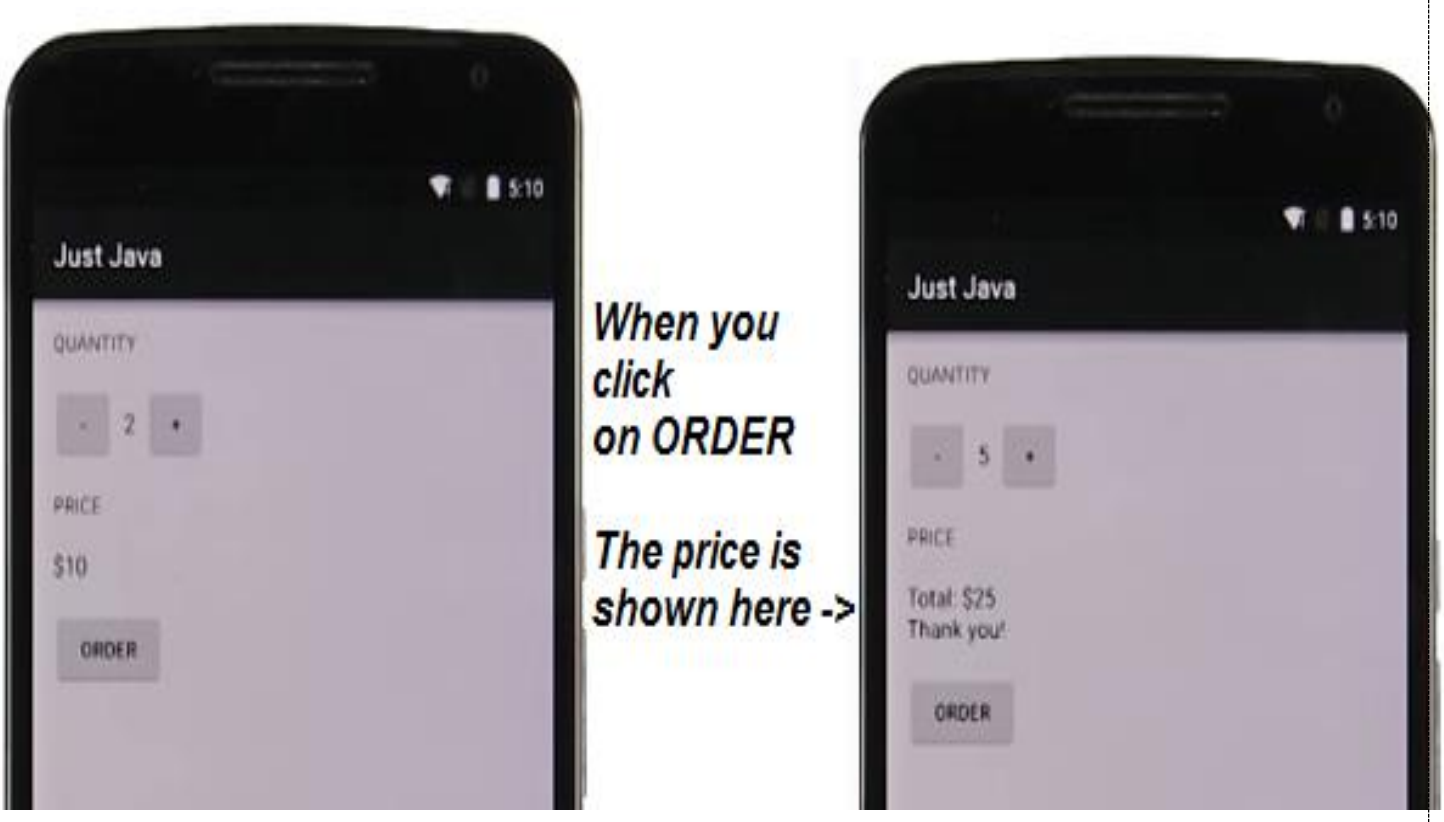
Calling a method

- ☐ Call `calculatePrice()` from `submitOrder(View view)`

```
int price = calculatePrice();
```

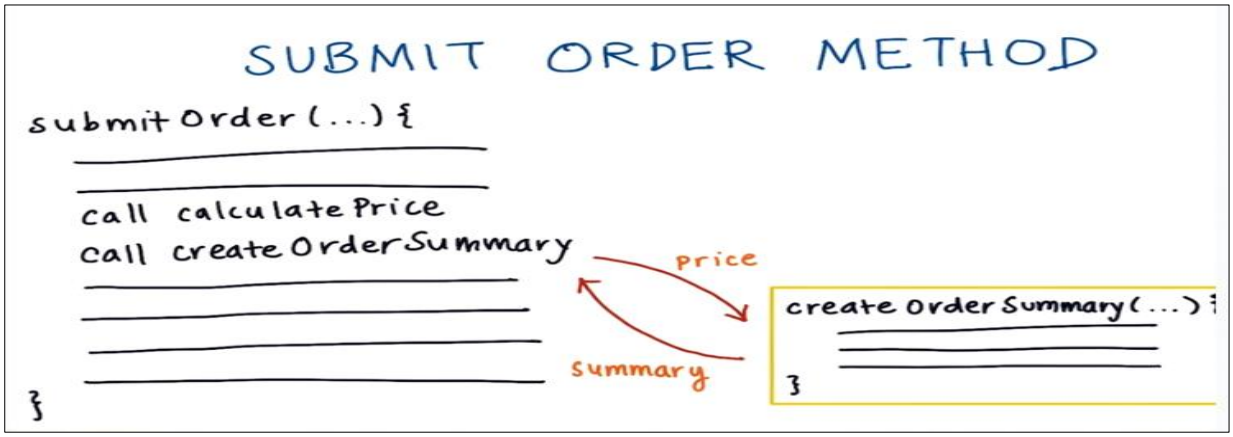
- ☐ Display price on screen.

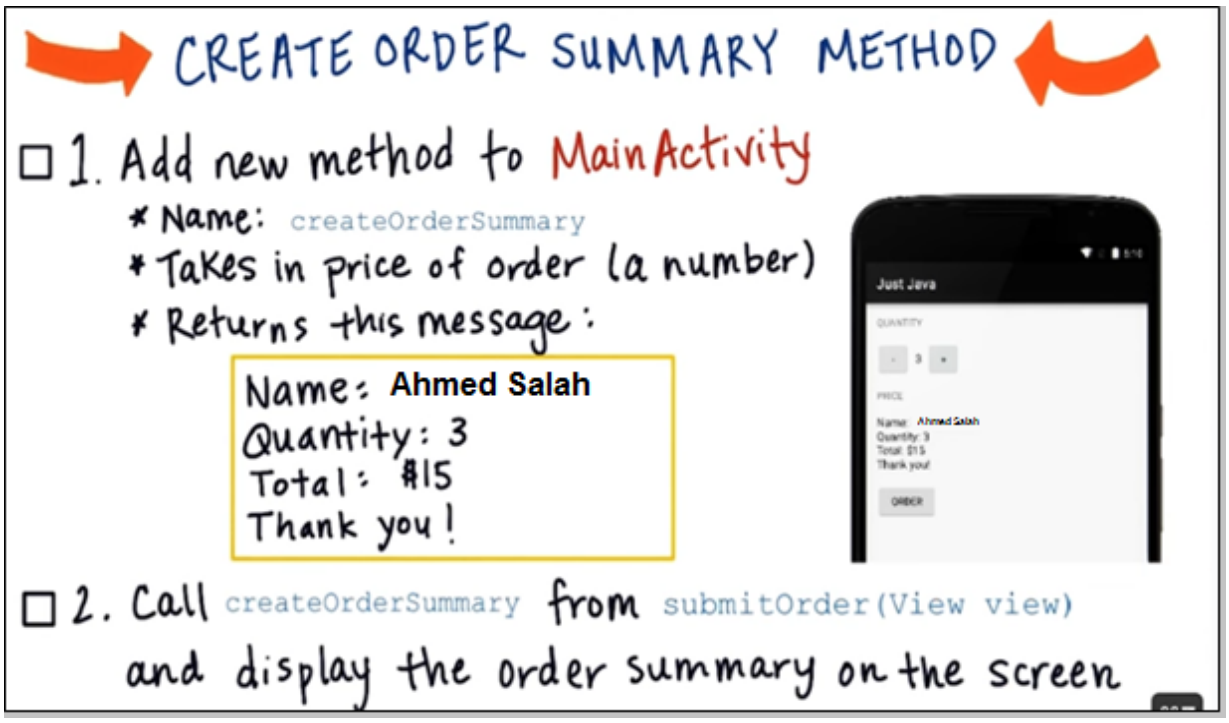
After you done you should get this result below:



Define and Call a Method

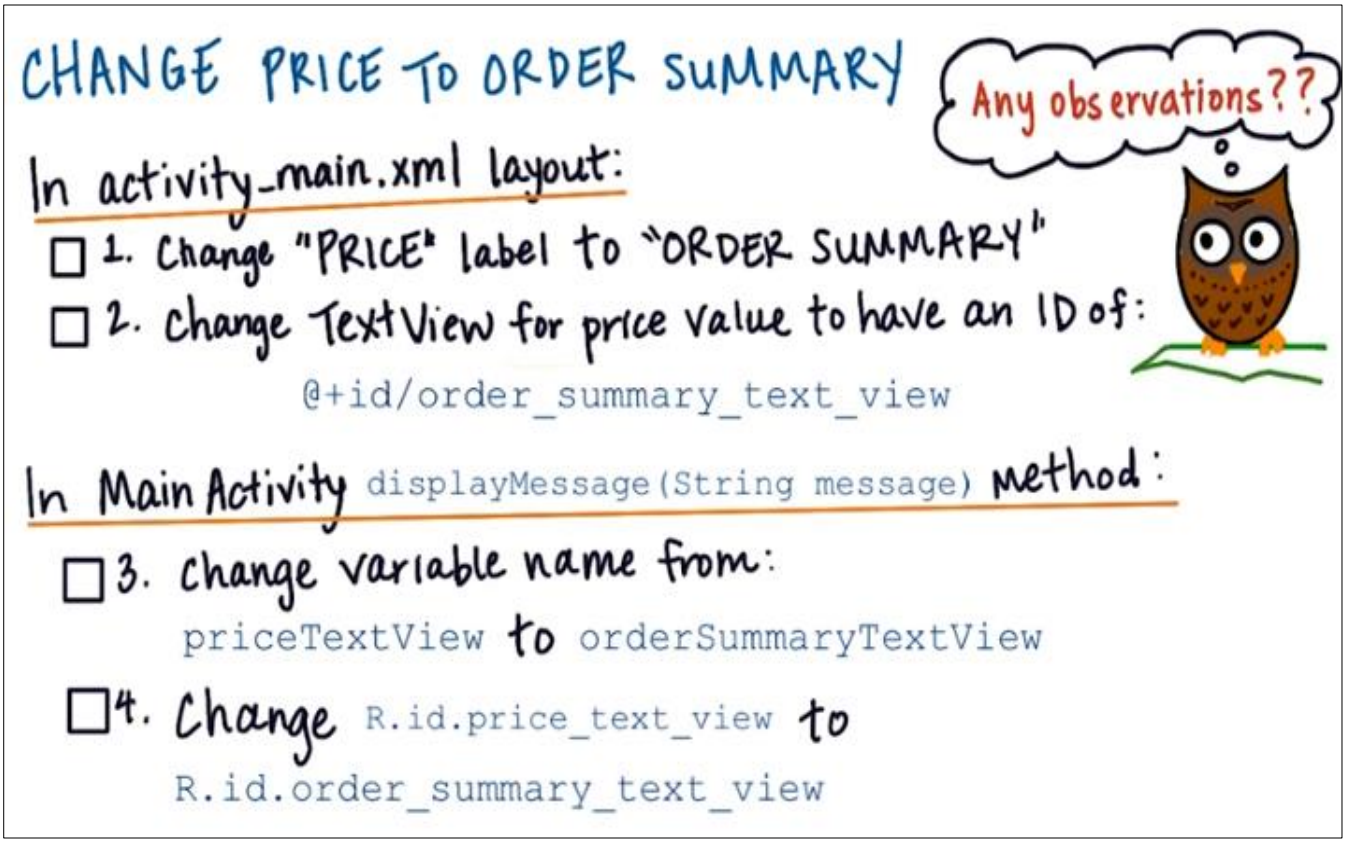
Now it's time for you to create your own method. The method should give order summary





The displayMessage Method

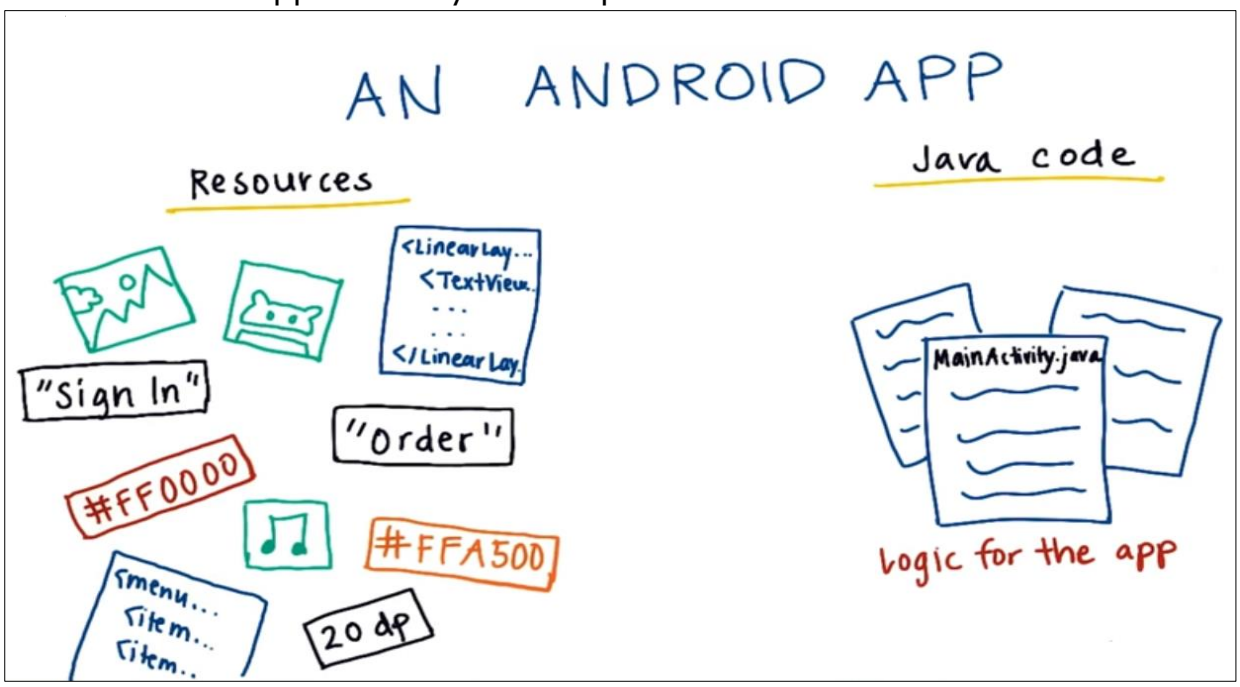
The end goal should look something like this:



In MainActivity.java: Remember to delete the displayPrice() method because we don't need that anymore.

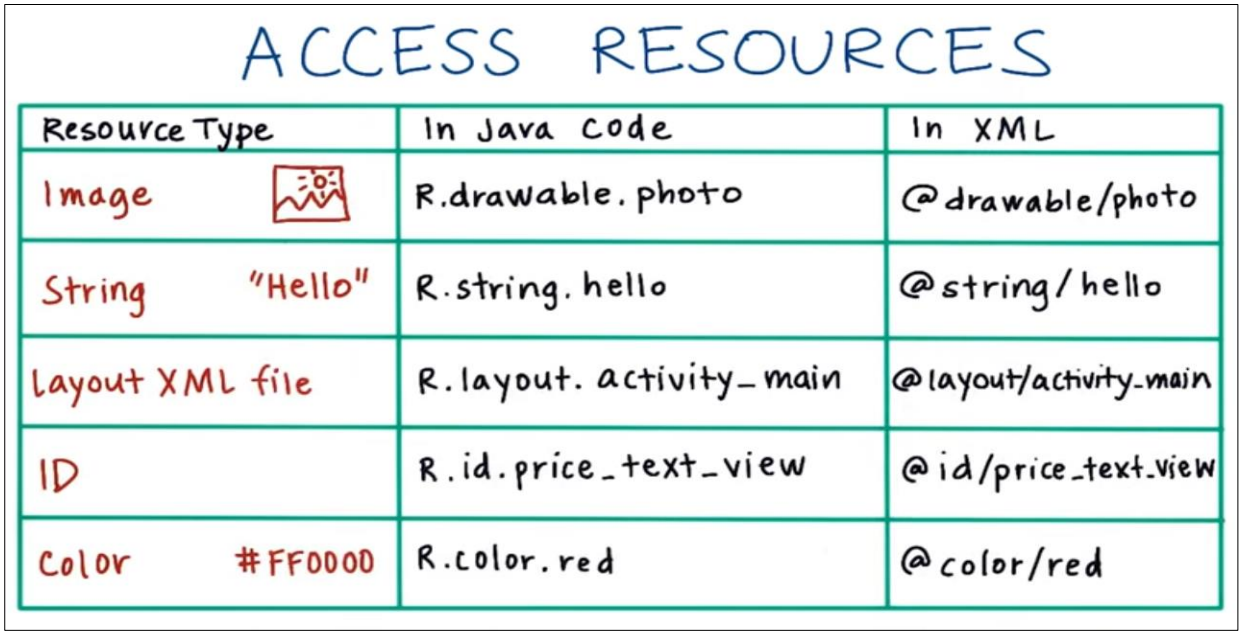
Resources

We are going to explain about resources in your App, resources are located in the **res** folder in your App, where java files are located into **java** folder and we have MainActivity.jave located inside this folder. Android App is mainly made up of Resources and Java Code.



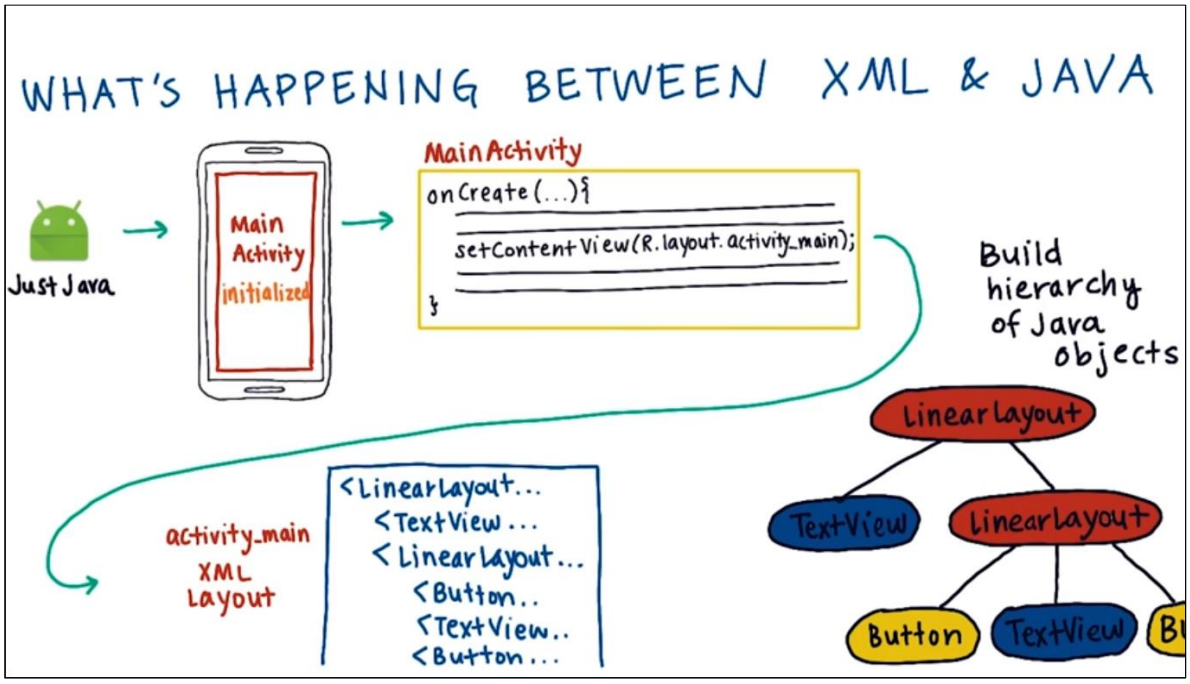
Resource IDs

Here are some of the most common resources but there are many others. Remember to use capital **R** to access them in Java. You can find resources by press **Ctrl+F** and search for **R.** with match case.



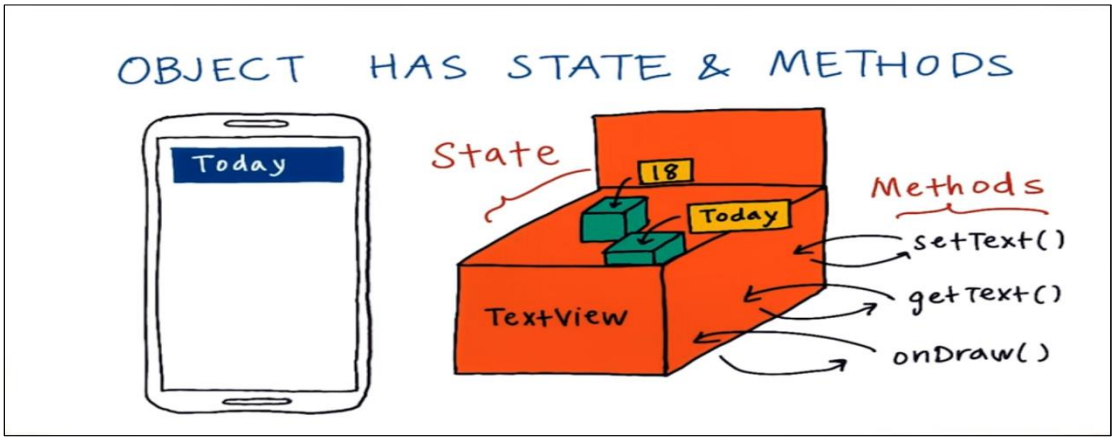
From XML to Java

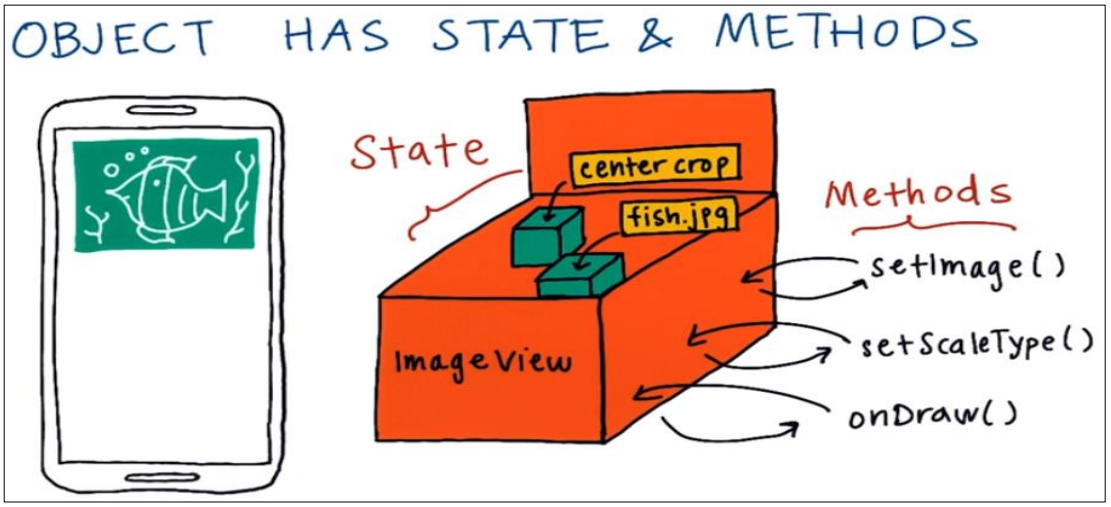
When you open click on Just Java App icon it opens MainActivity, now you don't see anything because it's still being initialized, in the Main Activity file the onCreate methods get call automatically and this method does a bunch of work, then we have a line that says **setContentView** and then the resource ID for the layout file. This mean that the content view of the activity should be set to the specified layout in the XML file. One you set the resource layout that you're going to use for content view, then the android device can go ahead and start parsing this XML layout file, then it inflates a Java object to represent that linear layout, then it goes to next line and see that we need a TextView as a child to that linear layout. What I didn't tell you before is that this is actually a hierarchy of Java objects. And the process continue in the same manner with next LinearLayout and TextView. Once we have this view hierarchy of Java objects, then we can manipulate and change them while the app is running. An object like TextView here contains state information like the text , textColor, and textSize the object has also methods so we can call these methods and change their internal state. So for example we can change the text while the app is running. Interacting between all these objects is called object oriented programming. To summarize, when we call setContentView to this layout, then we read this layout and then inflate a bunch of java objects. These Java objects make up the view hierarchy of the App. The MainActivity holds on to this view hierarchy and then as you interact with the MainActivity, it can updates these objects



What Are Java Objects?

To show TextView is actually controlled by a Java object within the app. This is an example of TextView Java object, it's like a box and inside of it, and it holds state. The state is made up of a bunch of fields, a field is a variable. We can have a text field to store the today string which is going to be the text displayed on this screen, we can also have an integer field store the font size of this text. There's also a bunch of other fields like text color, font family that make up the state of the TextView. To interact with this Java object and change it while the app is running we can call methods on the text TextView. Some example methods on a TextView are setText, getText, setTextSize, getTextSize. Each of these methods just executes a list of instructions. You can have more complex methods such as onDraw which handles drawing TextView on the screen

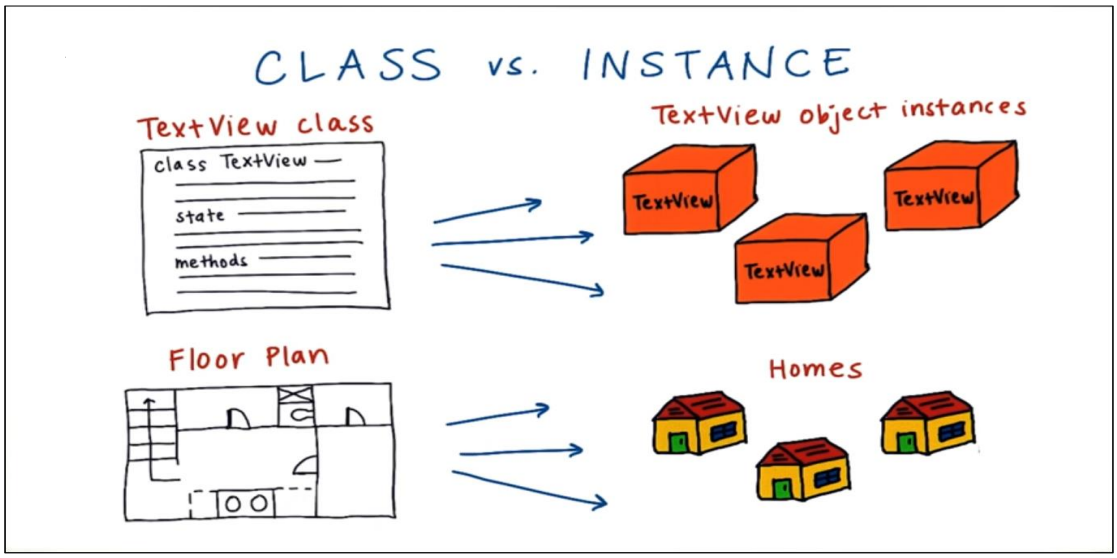




So how to know what are all the fields inside the state of an ImageView and what all the methods are of this ImageView?

To answer that question we have to look at the class definition for that object it's a file called TextView.java similar to how we defined MainActivity.java. Inside this class TextView file, we have a bunch of

codes that talk about the state of the TextView as well as the methods. Now it doesn't say the state and methods exactly, but it just contains information on the states and methods. The TextView constructor inside the class is used to create instances.



Below is a **simplified version** of TextView class to make it clearer.

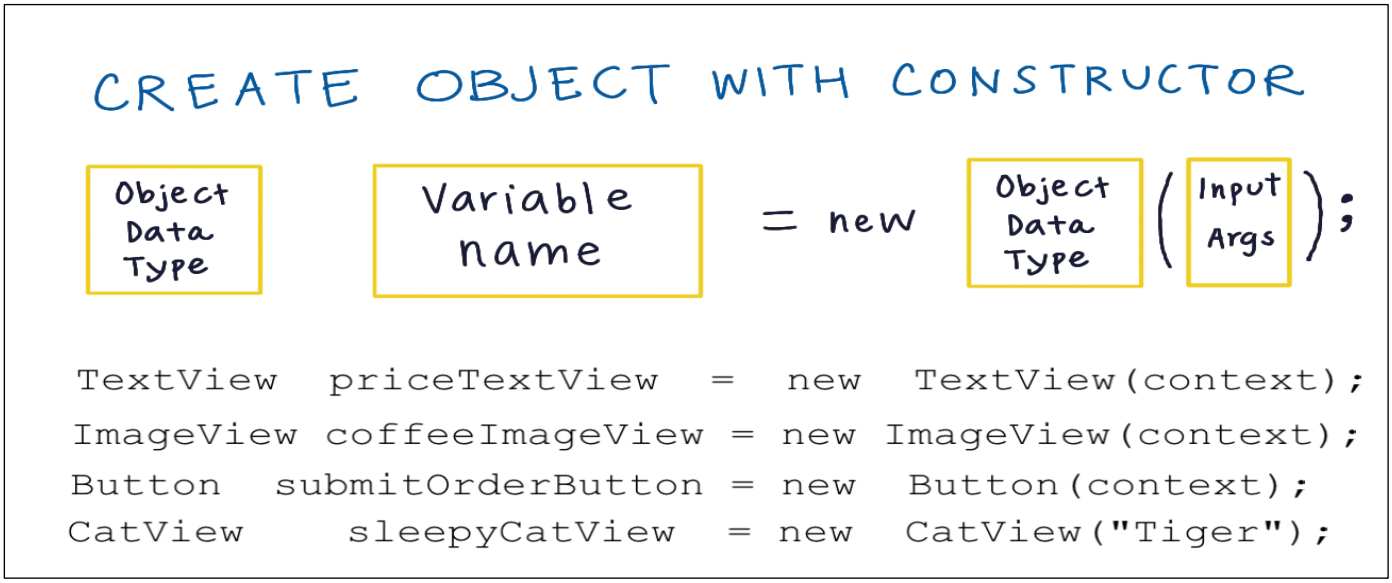
TextView.java

```java
/**
 * Displays text to the user.
 */
public class TextView extends View {

    // String value
    private String mText;

    // Text color of the text
    private int mTextColor;

    // Context of the app
    private Context mContext;

    /**
     * Constructs a new TextView with initial values for text and text color.
     */
    public TextView(Context context) {
      mText = "";
      mTextColor = 0;
      mContext = context;
    }

    /**
     * Sets the string value in the TextView.
     *
     * @param text is the updated string to be displayed.
     */
    public void setText(String text) {
        mText = text;
    }

    /**
     * Sets the text color of the TextView.
     *
     * @param color of text to be displayed.
     */
    public void setTextColor(int color) {
        mTextColor = color;
    }

    /**
     * Gets the string value in the TextView.
     *
     * @return current text in the TextView.
     */
    public String getText() {
        return mText;
    }

    /**
     * Gets the text color of the TextView.
     *
     * @return current text color.
     */
    public int getTextColor() {
        return mTextColor;
    }
}
```

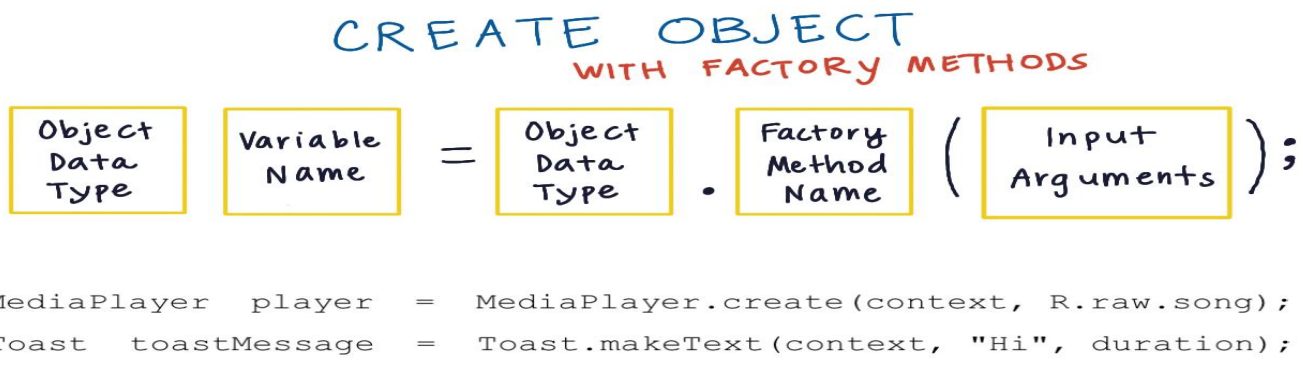
Small **m** letter means a member of the class of a field of the class.

Create an Object

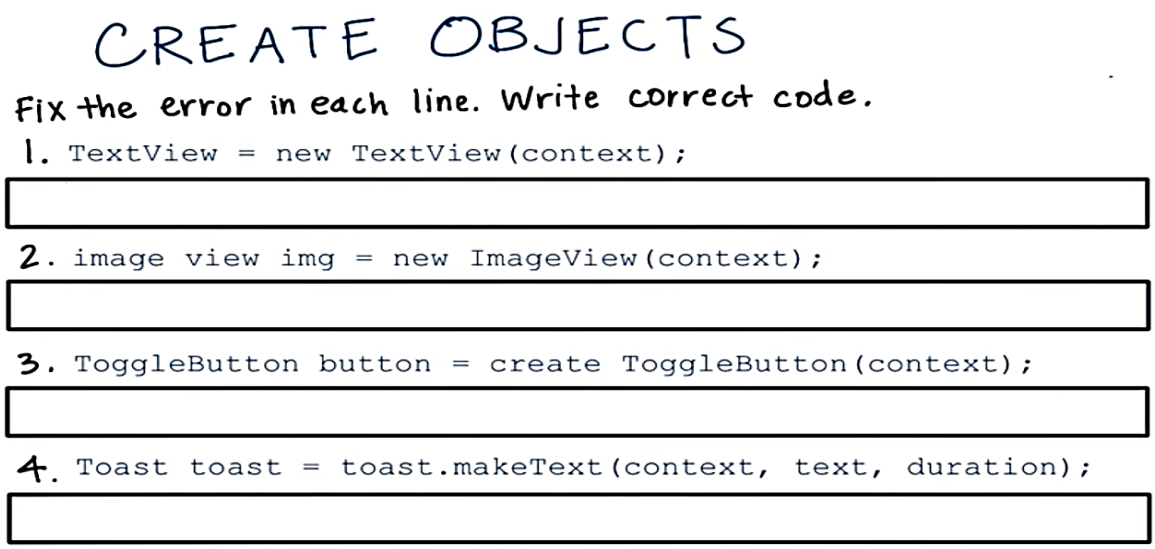
In last page we mentioned that the constructor is defined within the Class and it's used to create objects instances , so we Call Text View to create first instance , then we call again to create next one and so on. In these examples below we are creating different objects in storing them in variables that could store these object instances, then we use **new** and **constructor name** to call constructor to create new objects , we have parentheses and then we put the input in there. We can even define your own objects data type like you create a class definition for CatView, in this case the constructor requires an input string for name of the cat so that's why I put Tiger. To determine what input you should pass to these constructors you have to look at either the class Javadoc file or the class source file.



Sometimes you may face some objects that use a factory method to create an object that is instead of a constructor. Here is an example to create new MediaPlayer we use **MediaPlayer.create** and this returns an object that can be stored inside player variable according to format below.

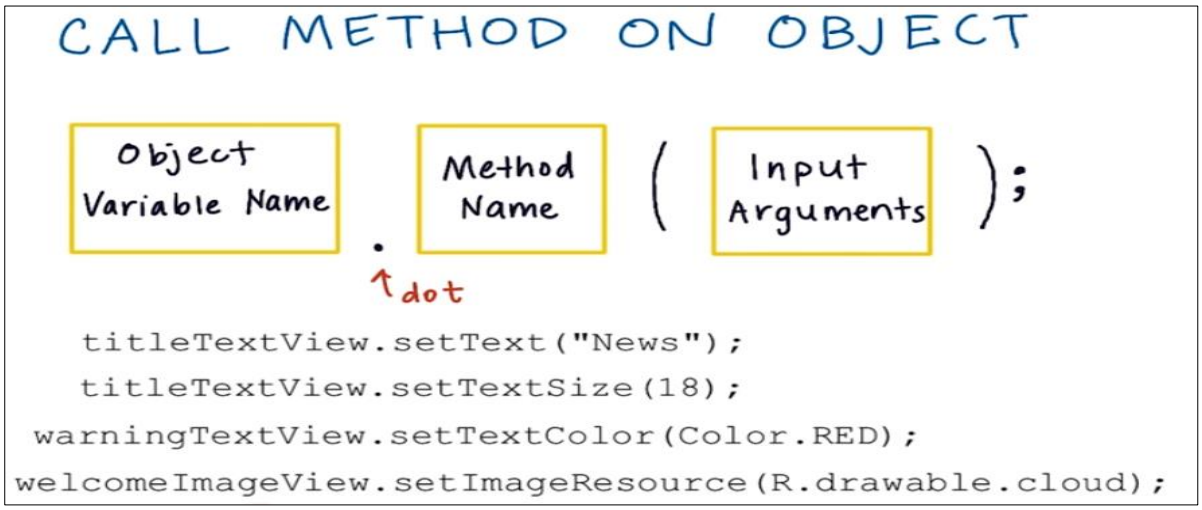


Now fix these errors according to last page example.



Call Methods on Object

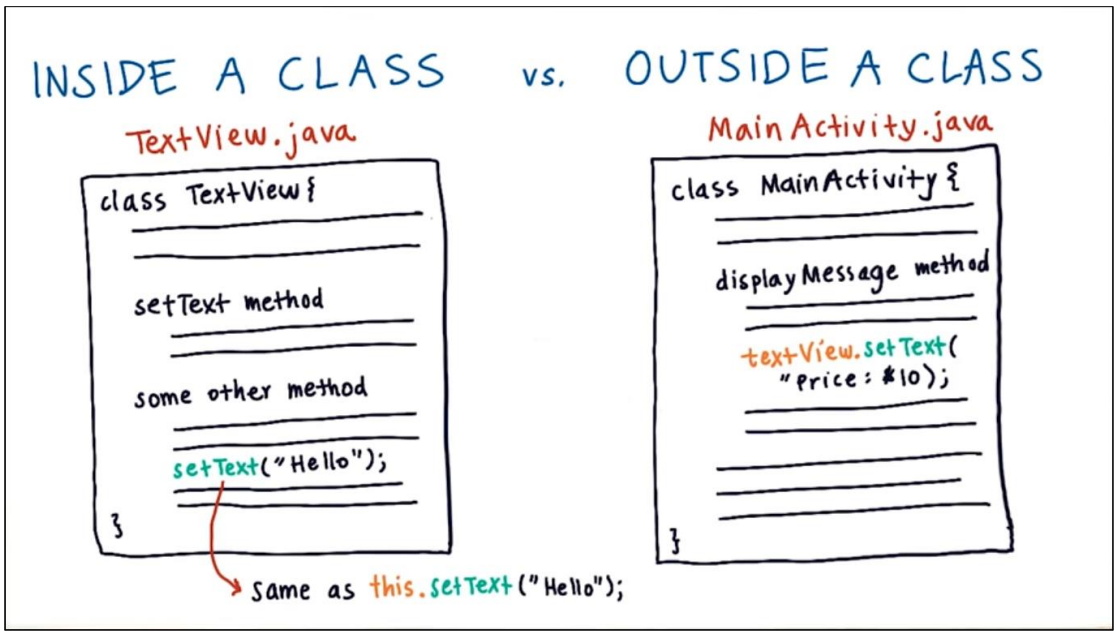
When we call a method on object is should follow this format in the figure. Please make sure that you always call the correct variable on the method because you might have many TextView variable on your App.

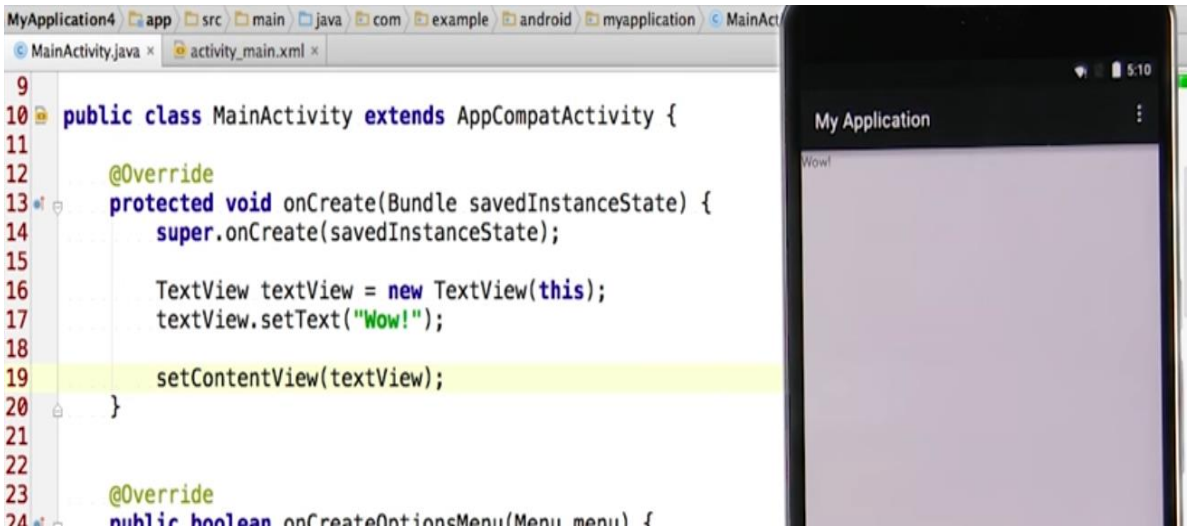


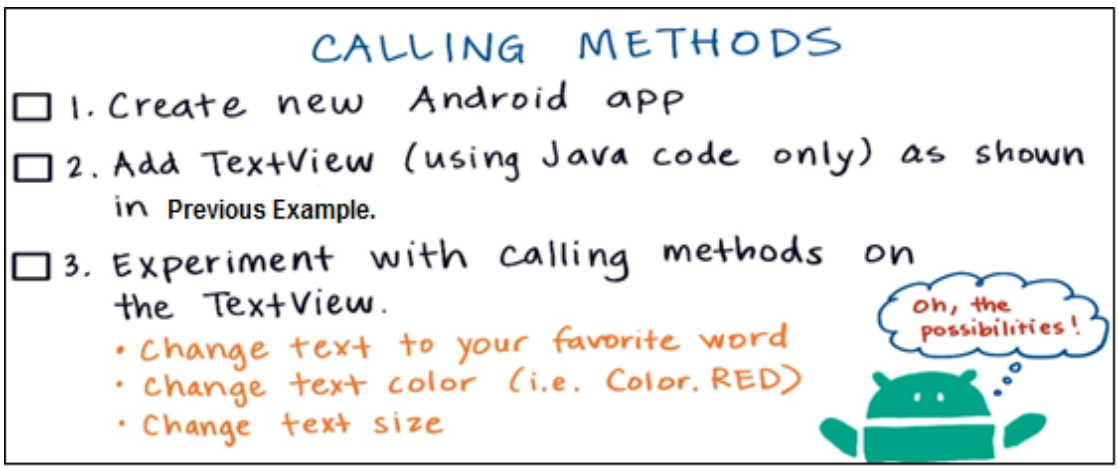
To get full list of methods that you can run on TextView object, then open Android documentation page by googling it "textview android".

Calling method such as setText inside the class you don't need to use class name, while using a method outside that calls requires you to mention the variable name to decide which TextView need to change in case you have many text views inside that class.

Another difference is within the TextView class you can access private and public methods, but if you are outside the TextView class you can't use private methods and variables.



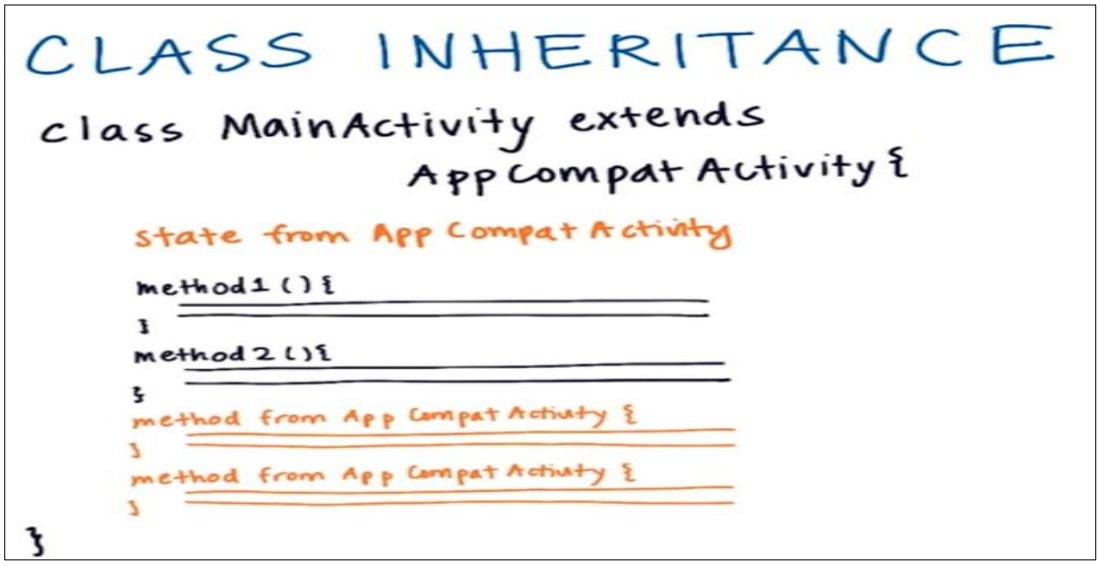




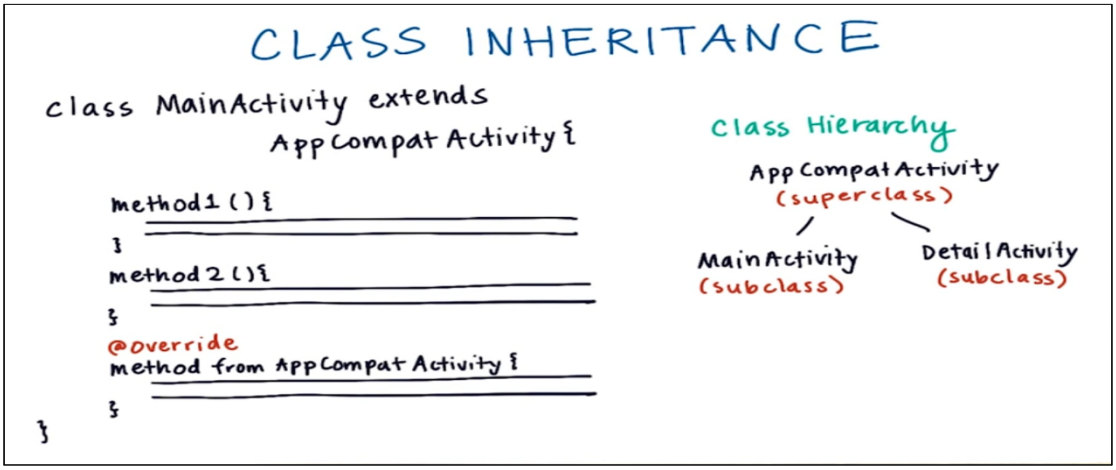
Inheriting Behavior

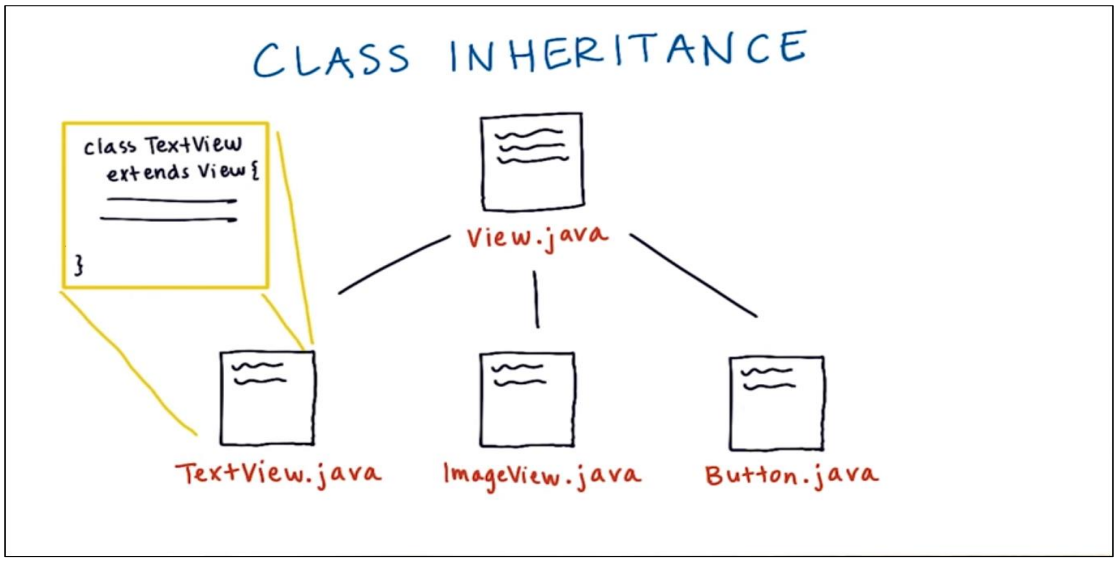
If you look at the definition of findViewbyId(), setContentView inside the MainActivity.java , you will not find their declaration.

You might notice the MainActivity extends AppCompatActivity, this means that the main activity is an extension of the functionality in the AppCompatActivity class and it's a part of the Android Support library and it allows us to use the latest UI features on Android while still working on old Android devices. By extending AppCompatActivity class we are getting all the functionality, all the states, methods from here within the MainActivity for free, we don't have to copy and paste any code from that class we just simply extend that class.

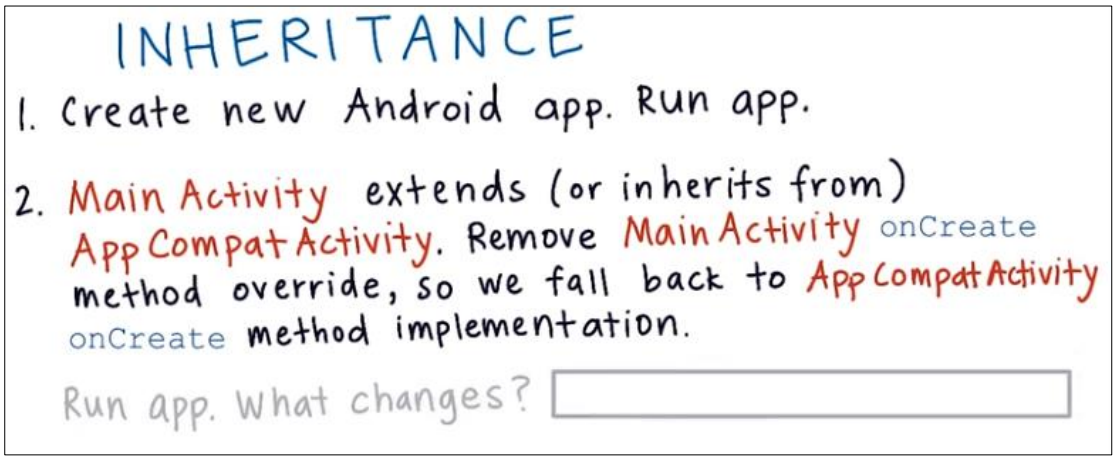


Regarding inheritance things there's good and bad parts about that. Sometimes we might want to inherit behavior, but other times we want to slightly modify it. If you want to change some of the behavior from this inherited class, you can override certain methods. You can add this @override text on top of a method so that the computer knows that you don't want the version of method from the AppCompatActivity but you want this version that you have defined.

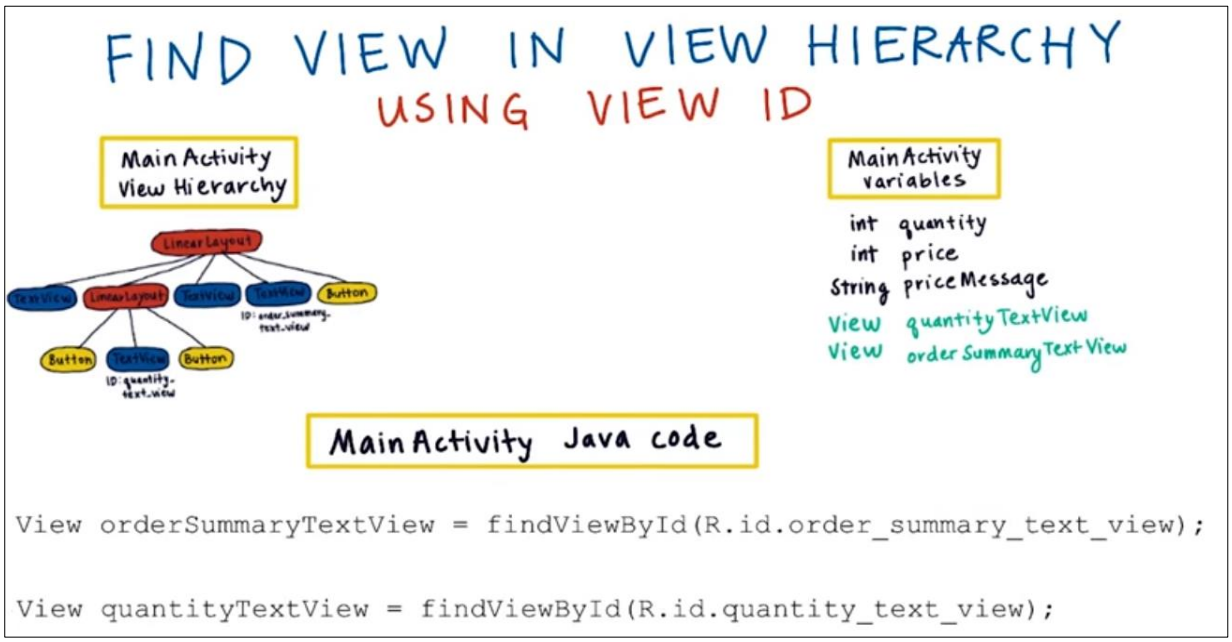




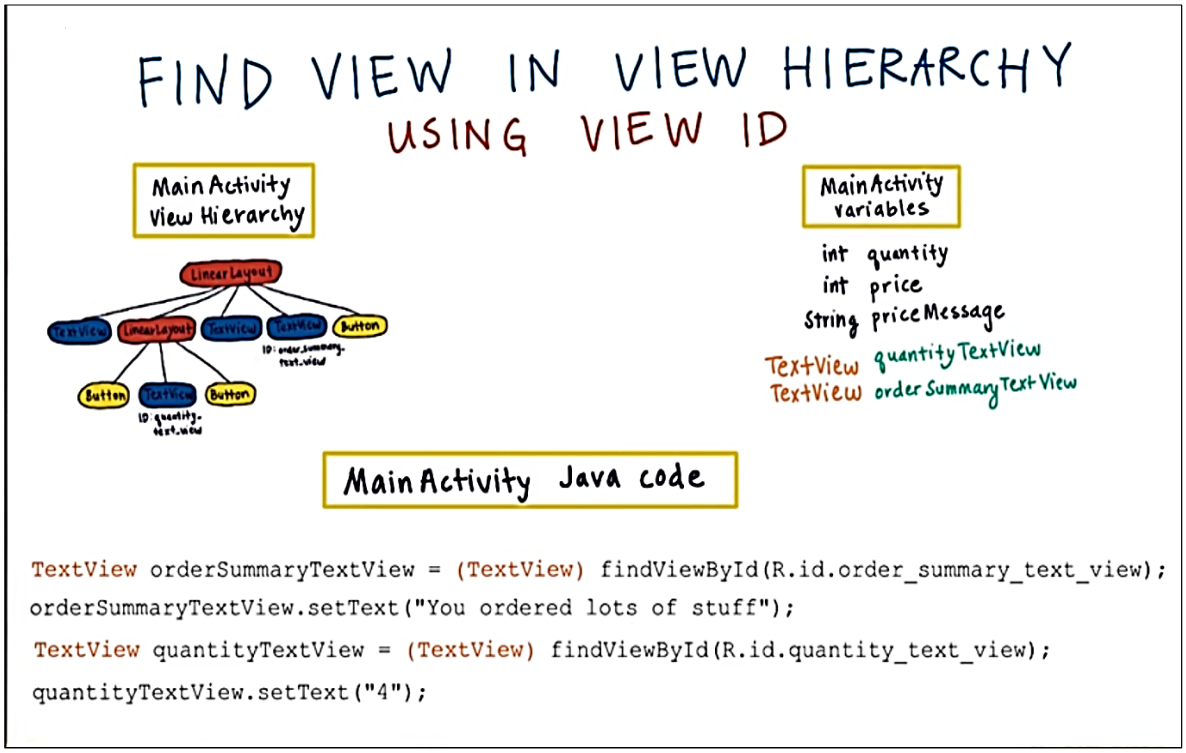
You can try this quiz to better practice.



Find View by Id

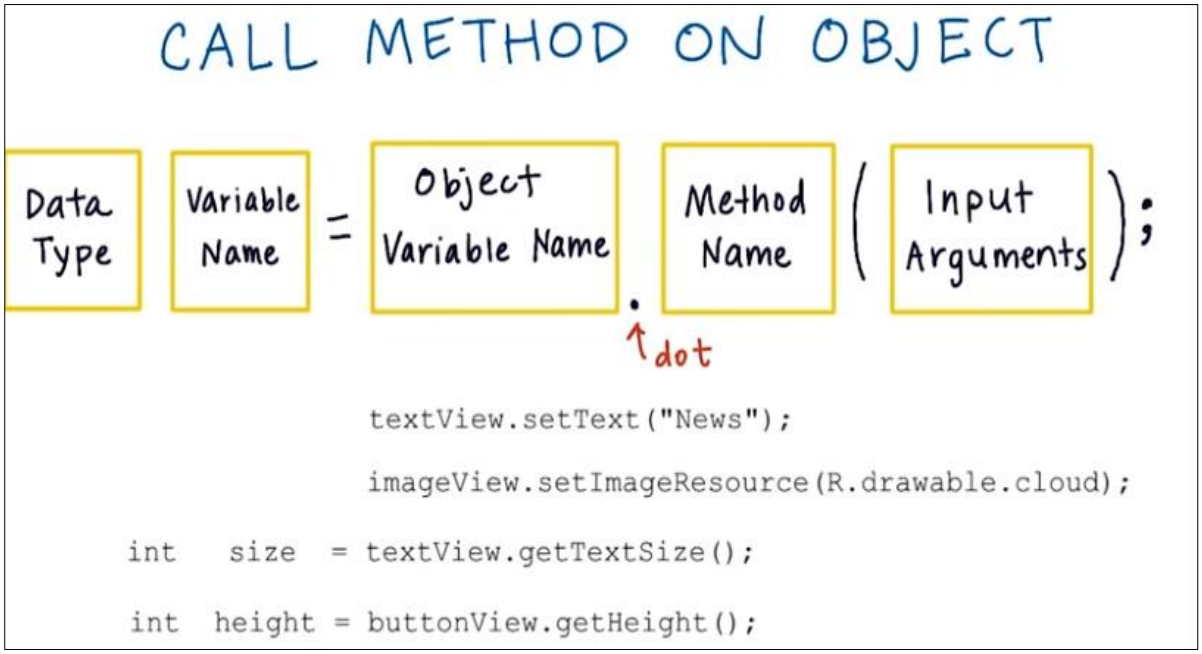


If you apply the two codes of Views in example before, you will get an error, because we didn't define the particular type of views which is TextView and there is no casting for these variables to store them in their variables as TextView.

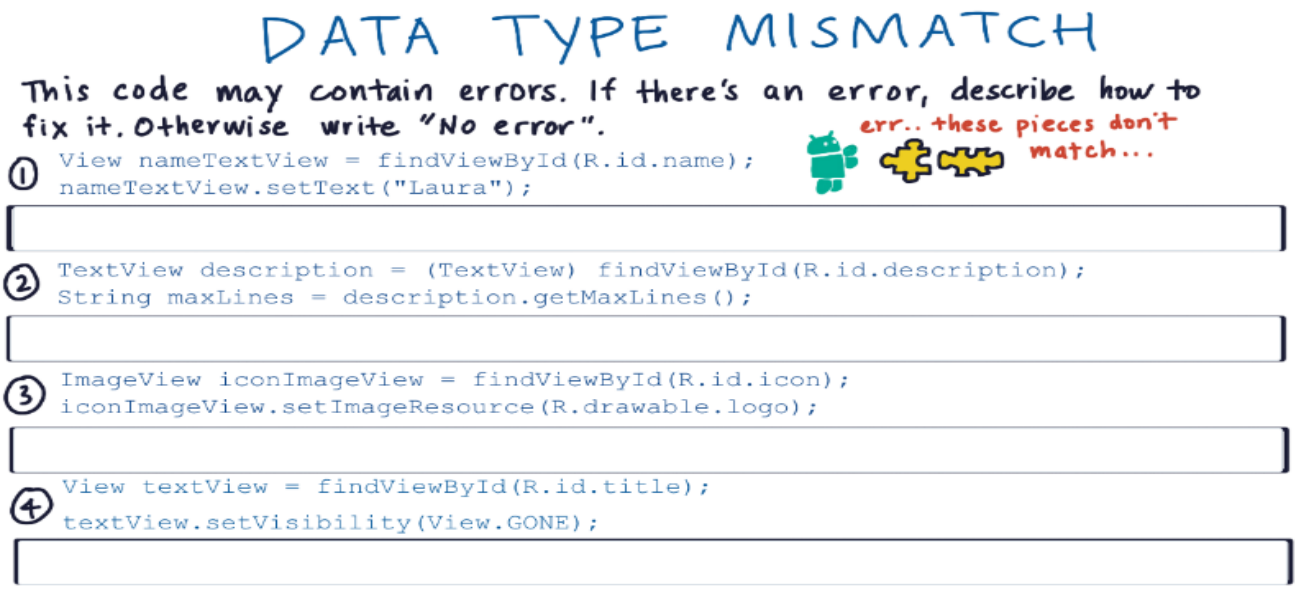


Casting with findViewById

For the example below we should make the return type as int because these methods return **int** data type, but we can't define return type as a string since these methods are returning **int** values.



Try next quiz to test your knowledge.



Set Data on Views

A Review Of What We Know

So we've now seen a few examples of how we can use the `setText` method to modify a view. Here's one example we just looked at:

```
TextView orderSummaryTextView = (TextView) findViewById(R
.id.order_summary_text_view);

orderSummaryTextView.setText(message);
```

You can break this down into two steps:

Step 1 : Get the view object using the view ID
The first line of code is getting the TextView and storing it in a variable named `orderSummaryTextView`. To actually get the view, we use the `findViewById` method which is a method in the **Activity class**. The argument required is the view ID, which we supply by typing `R.id.IDOFVIEW`. In this case, the ID of the view is order_summary_text_view, as set in the XML. Note that you need to **cast** the object, which is what the `(TextView)` is doing. It is saying that the value findViewById is specifically a TextView and not just a generic View.

Step 2 : Call a method ON the view object
Since we are calling a method **on** an object, we use the dot syntax. `orderSummaryTextView.setText(message);` is equivalent to saying, take the **orderSummaryTextView** object which has the capability to **setText** , and setText to whatever string it is passed (in this case the string variable **message** is passed).

Your Turn

You're going to practice these steps—getting the view object, storing it in a variable, and then manipulating that view object.

Step 1: Create a New Project

To start this exercise, create a new project (use the **Empty** Activity template on Android Studio 1.4 and newer). The application name should be **Cookies**. Create this new project the same way you created a project for the **NewYear** app, **Just Java**, and **Court Counter**.

Step 2 : Copy over files

Copy over the Java and XML code into the correct files. The code is below. Then you'll want to copy the line below into your app's build.gradle file:
compile "com.android.support:appcompat-v7:22.1.0"

Android studio may suggest a newer version of the appcompat library, and you should use that one.

Also place these two image files in the drawable folder, as you did in the **New Year app**. They can be downloaded from the **server**.

activity_main.xml

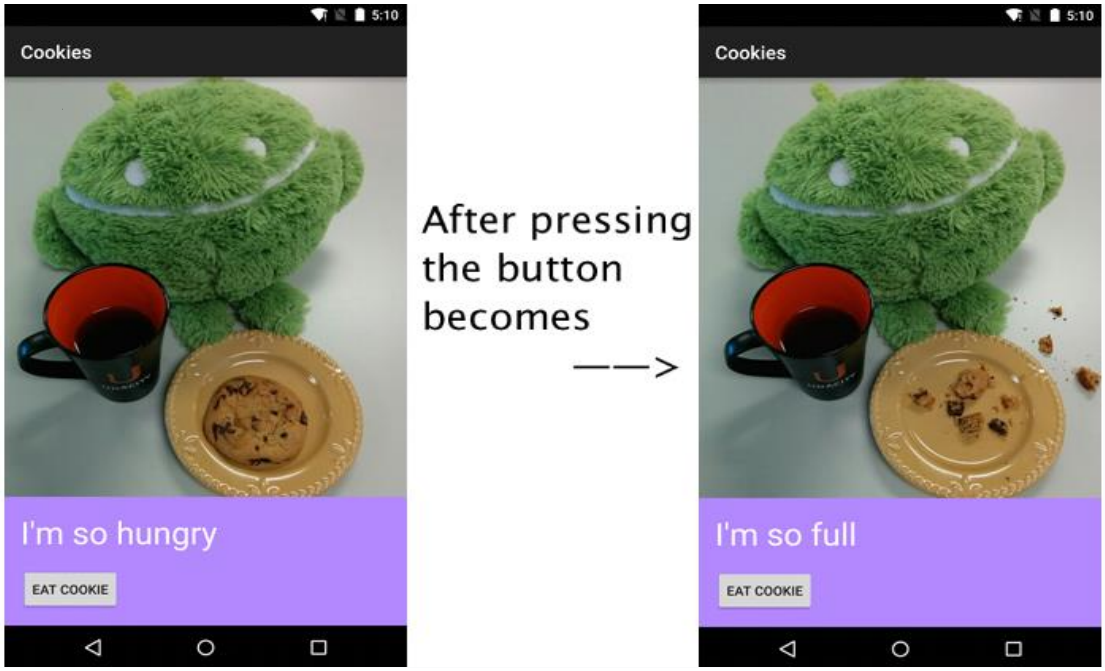
```
<LinearLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#B388FF"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/android_cookie_image_view"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:scaleType="centerCrop"
        android:src="@drawable/before_cookie" />

    <TextView
        android:id="@+id/status_text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginTop="16dp"
        android:text="I'm so hungry"
        android:textColor="@android:color/white"
        android:textSize="34sp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:text="EAT COOKIE" />

</LinearLayout>
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /**
     * Called when the cookie should be eaten.
     */
    public void eatCookie(View view) {
        // TODO: Find a reference to the ImageView in the layout. Change the image.

        // TODO: Find a reference to the TextView in the layout. Change the text.
    }
}
```

Step 3 : Hook up the button

Hook up the “Eat Cookie" button so that, when it's clicked, the image and the text change as seen below.



You’ll need to modify the XML to handle a button being pressed (you’ve done this in before). Then you’ll need to use the skills you just learned to manipulate the image and text with the Java code. Good luck!

Read Data from Views

Getter and Setter Methods Review

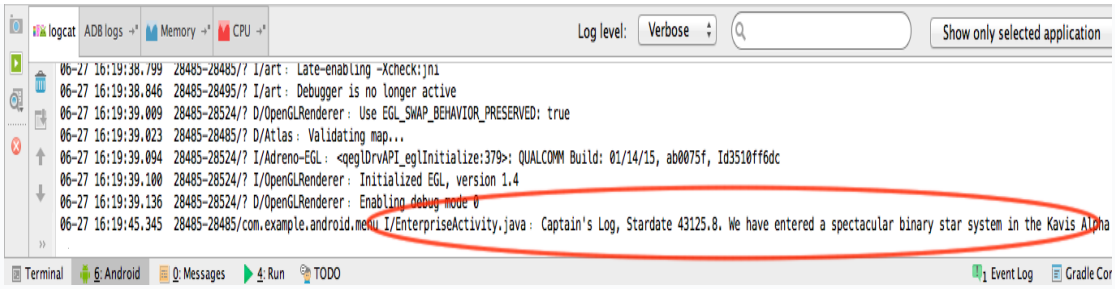
You’ve been using methods such as `setText` and `setImageResource`. These are called *setter* methods because they are meant to modify or manipulate one value of a view (such as the text or image that it stores). Conventionally they start with the word "set".

There’s also a category of methods called *getter* methods, whose sole purpose is to "get" one value of a view, such as getting the current text of a view. Conventionally they start with the word "get". We’ll be using some getter methods in this next exercise.

Logs

Another skill you will need for this exercise is the ability write to the Android Logs. More information can be found **here**, but essentially you write a Java statement like this in your code:

```
Log.i("EnterpriseActivity.java", "Captain's Log, Stardate 4
3125.8. We have entered a spectacular binary star system in
the Kavis Alpha sector on a most critical mission of astrop
hysical research.");
```

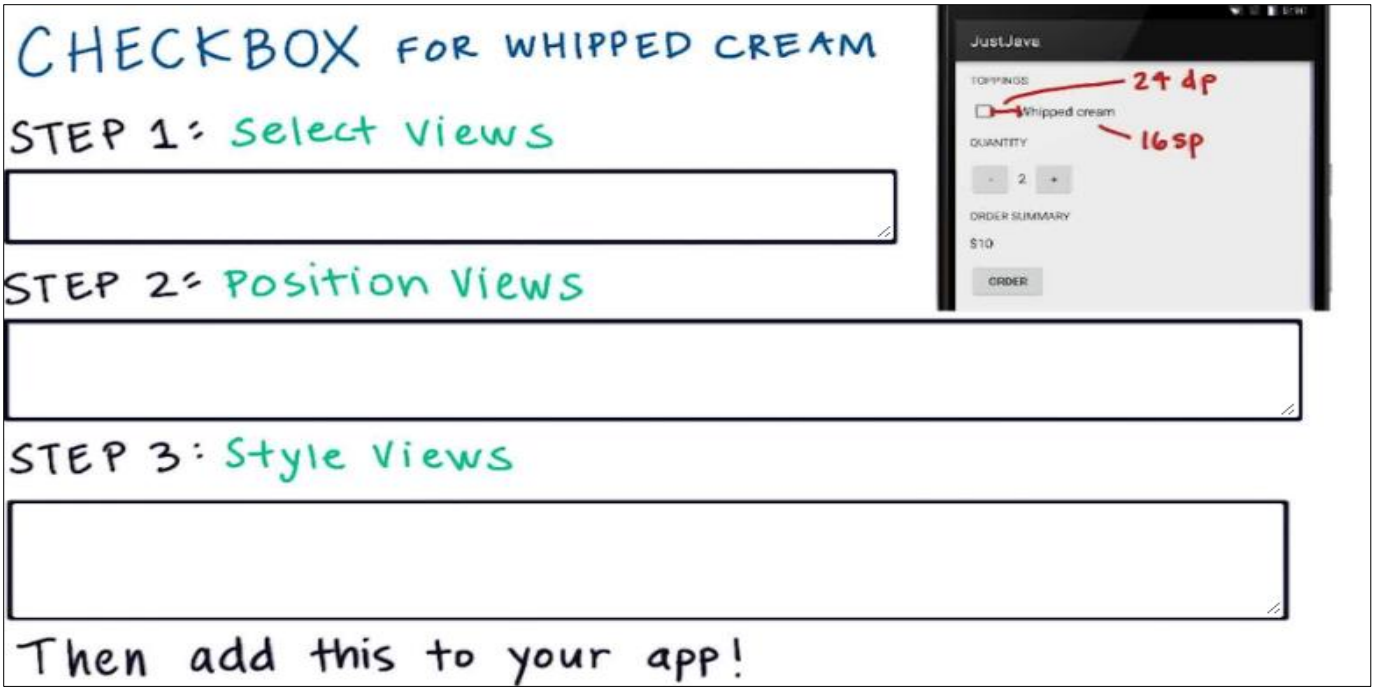


Note the first argument is the **name of the class** that the logging statement comes from. The second is the **text** you want to display.
We've used Log.i() here which stands for an "information" level log. You have these other options as well:

- **e(String, String)** (error)
- **w(String, String)** (warning)
- **i(String, String)** (information)
- **d(String, String)** (debug)
- **v(String, String)** (verbose)

Adding a CheckBox

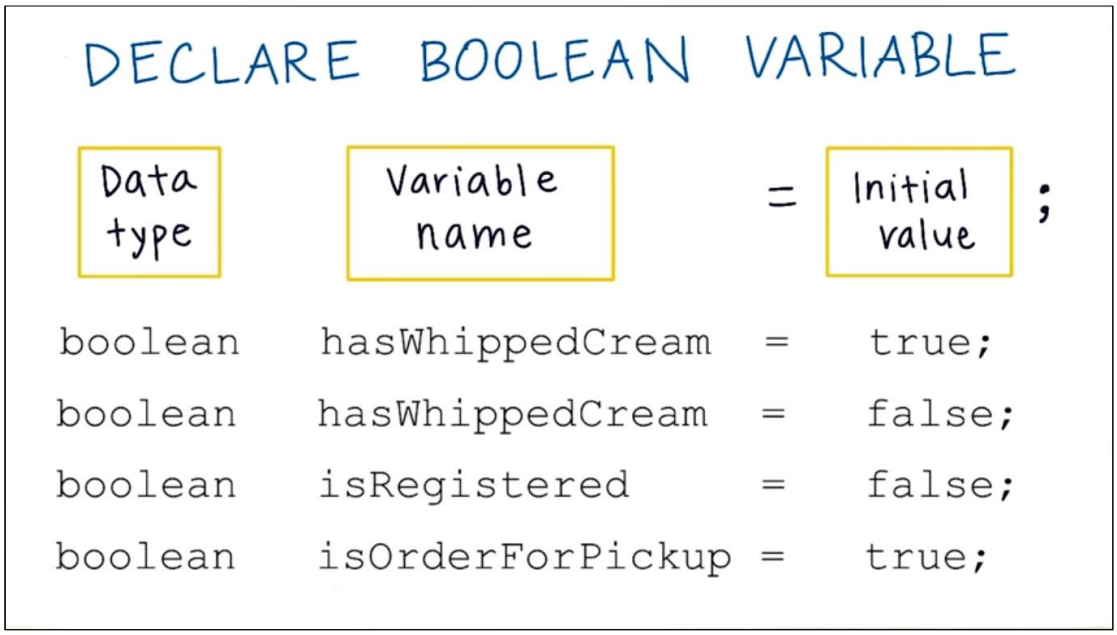
Now go to your JustJava app and add this CheckBox shown in example.



Boolean Data Type

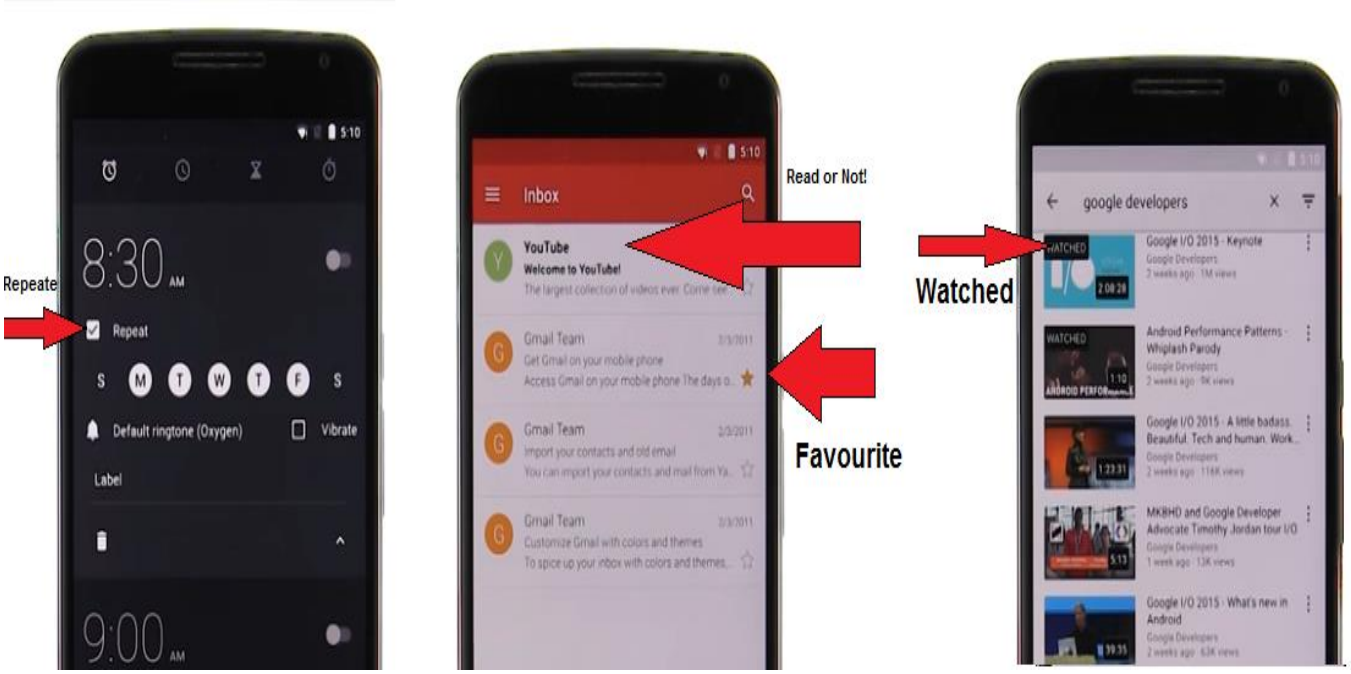
Now for the CheckBox we need a Boolean data type to determine the user have checked the box or not?

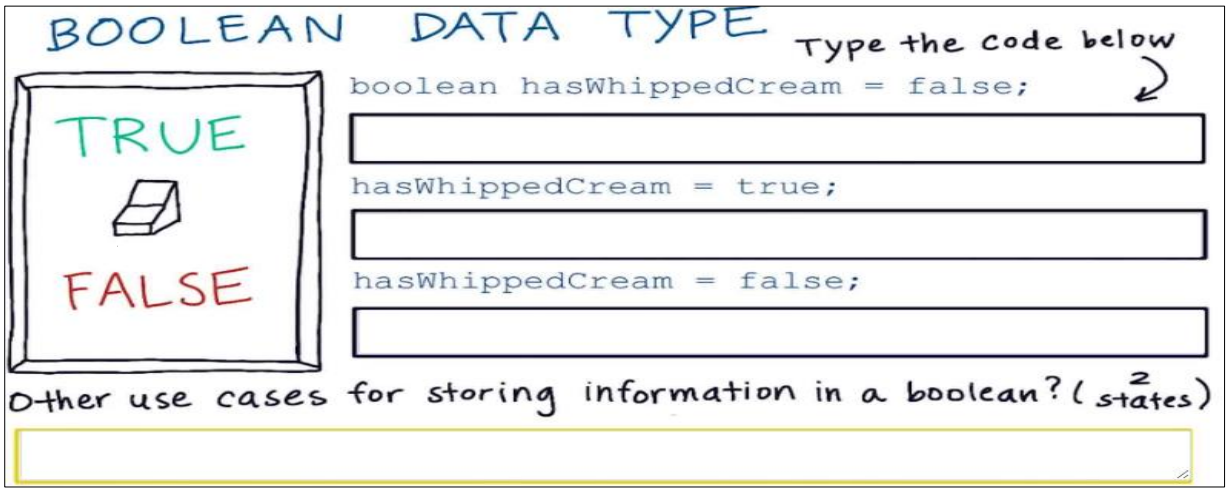
The Boolean data type has only two values true or false, and these true or false are literals but not strings. A Boolean variable suitable when we have two possible states, so when you have more than two states you can't use Boolean you might use string or integer.



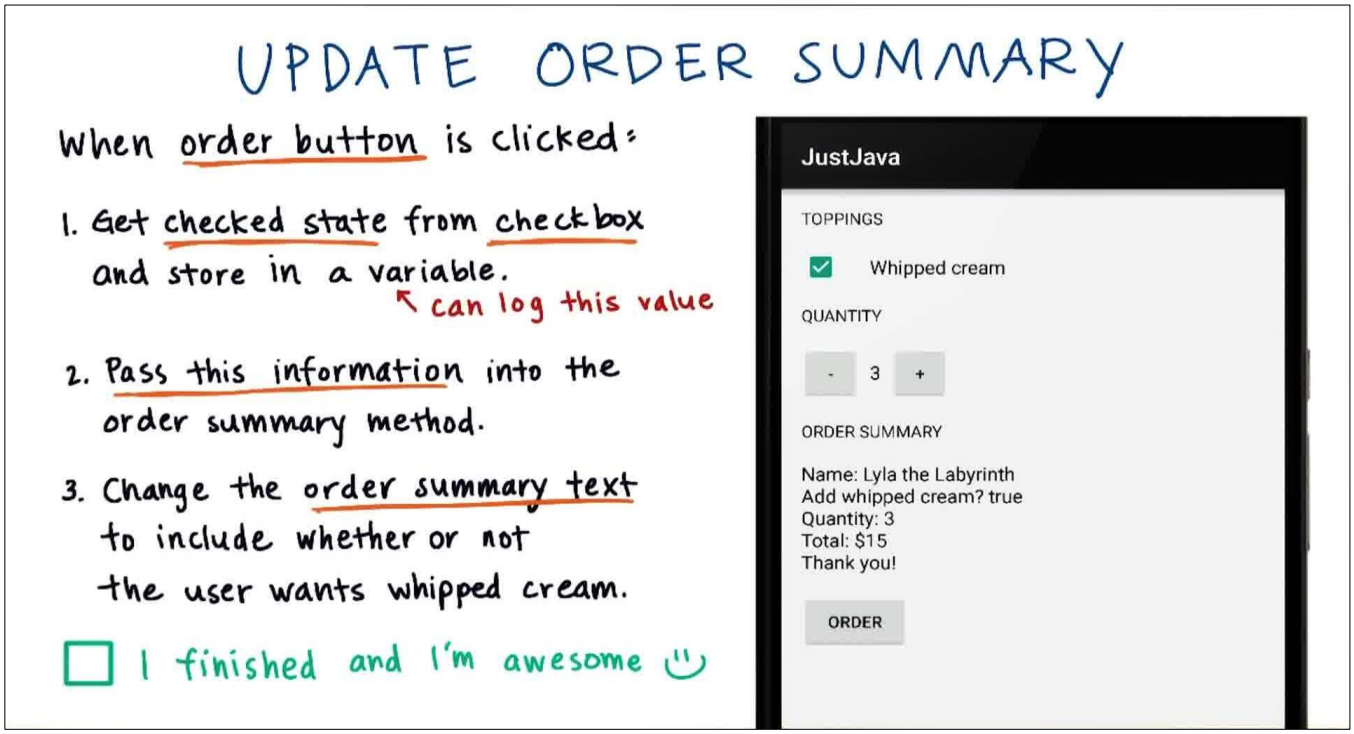
As you can see it's preferred to start Boolean variable name with has or is, so when you see such variable it's usually a Boolean but not always.

Below are some Apps where you can see Boolean data type such as Alarm, Gmail, and YouTube.





Update Order Summary

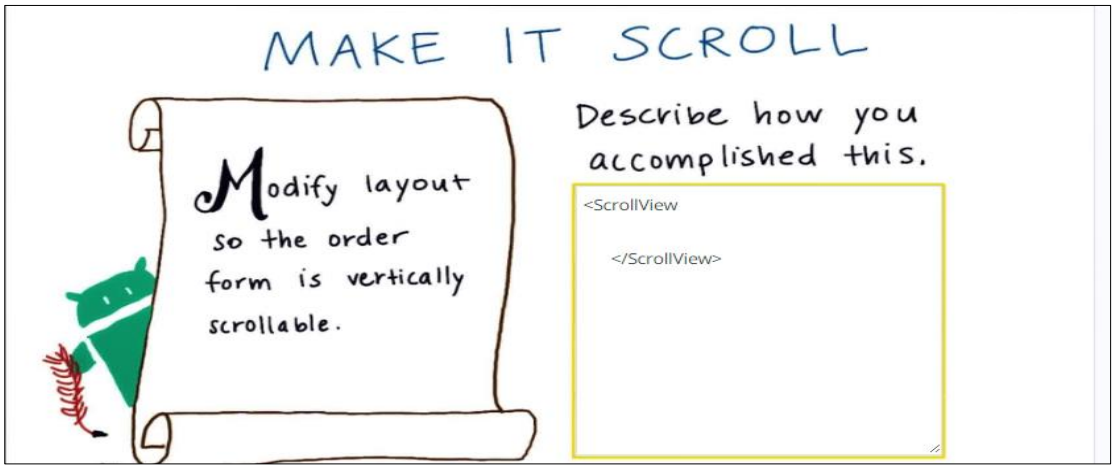


Here's some more detailed info for each step:

- When the button is clicked, find the CheckBox view, get checked state from the CheckBox, and store the checked state value in a new boolean variable. Feel free to add a log message to verify that step 1 is completed correctly.
- Pass the checked state boolean into the createOrderSummary() method, so it takes in 2 input parameters. The new input parameter is a boolean called `hasWhippedCream`. Remember to modify the method signature of the method.
- Modify the `createOrderSummary()` method so it displays this text on screen using the boolean input parameter.
- Remove log messages that were added for debugging purposes.

Scrolling Along

When you start to add too many things then the content can get cut off at the bottom. A vertical linear layout doesn't scroll if it extends beyond the edge of the screen you need to add something to your App to make it vertically scrollable. This is especially a common problem when you turn your mobile to **landscape** mode and the screen is shorter compared to **portrait** mode.



Add the Chocolate Topping CheckBox

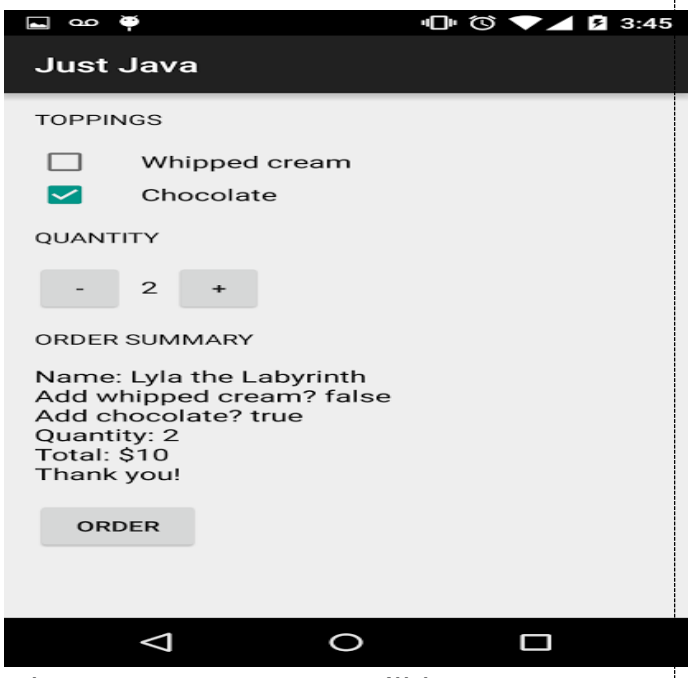
Things are getting tougher! We're offering less hand-holding, and letting you come up with the code yourself. In this task, you will implement a whole feature on your own.

Your Goal

Your goal is to add a chocolate topping checkbox in the Just Java app.

This will be similar to the **whipped cream** topping checkbox. This is what the app should look like when you're finished.

How to Start



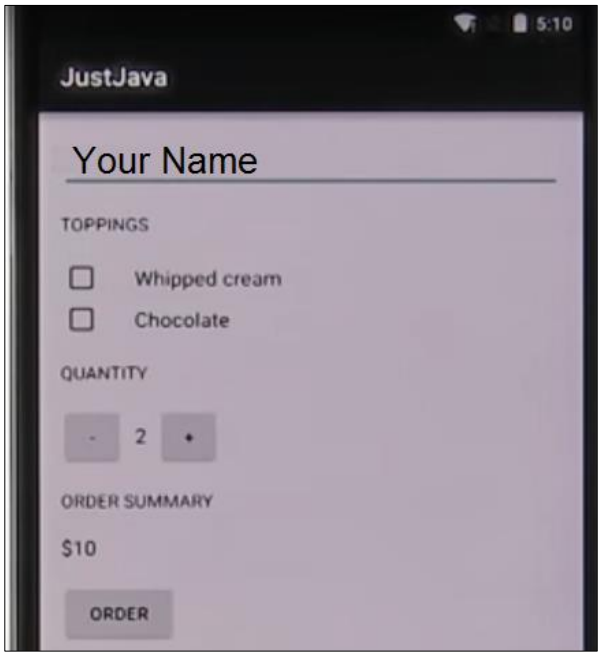
In the real world, you'll get requirements about what the app should do, and you will have to come up with all the steps to make the app reach that goal. To start, plan out the steps for how you will need to modify the code.

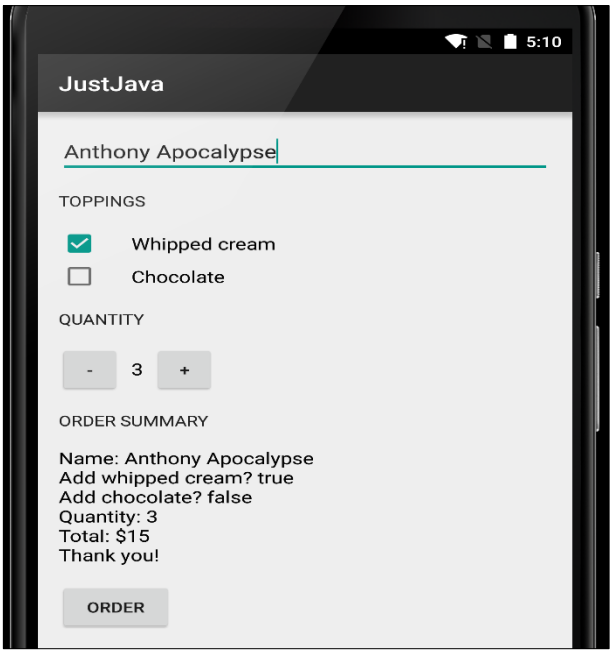
Every time you make a change to the code, stop and make sure your app still runs on your device and does what you expect. If you run your app after every change and you ever find an unexpected problem, it's obvious what change caused the bug. If you've made twenty changes since you last ran your app, you'll have a very hard time finding the source of any problems.

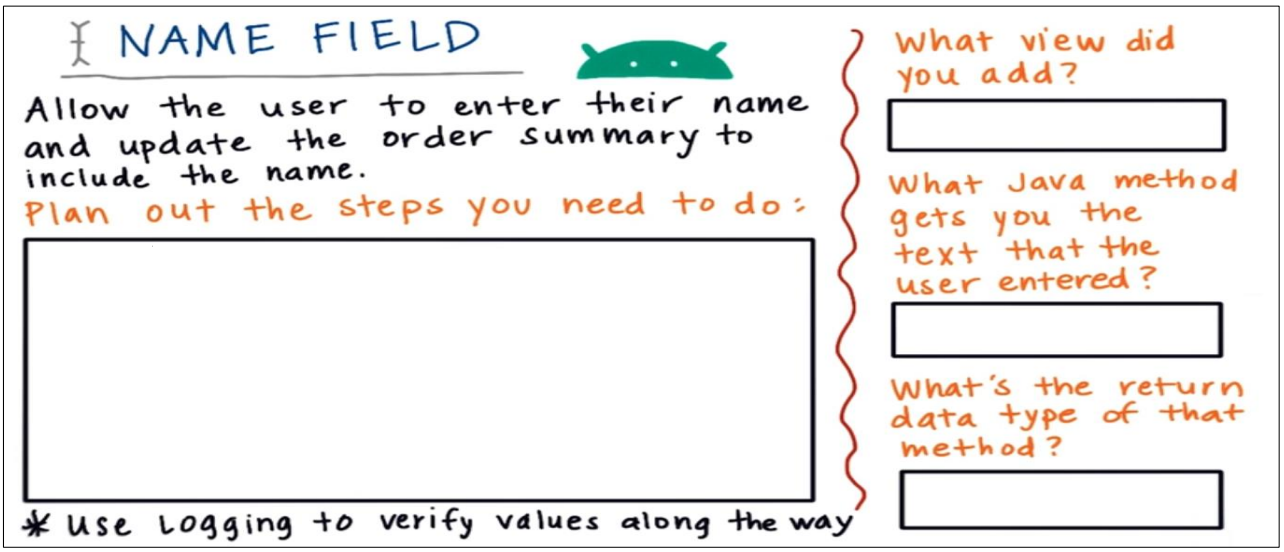
If you want to refresh your energy go and watch this **Googlers' Montage video**: https://youtu.be/pIvWpLgNWPA

What's Your Name?

At the end of this part, your App should looks like this one, and a user can enter his/her name by clicking in the field of text.



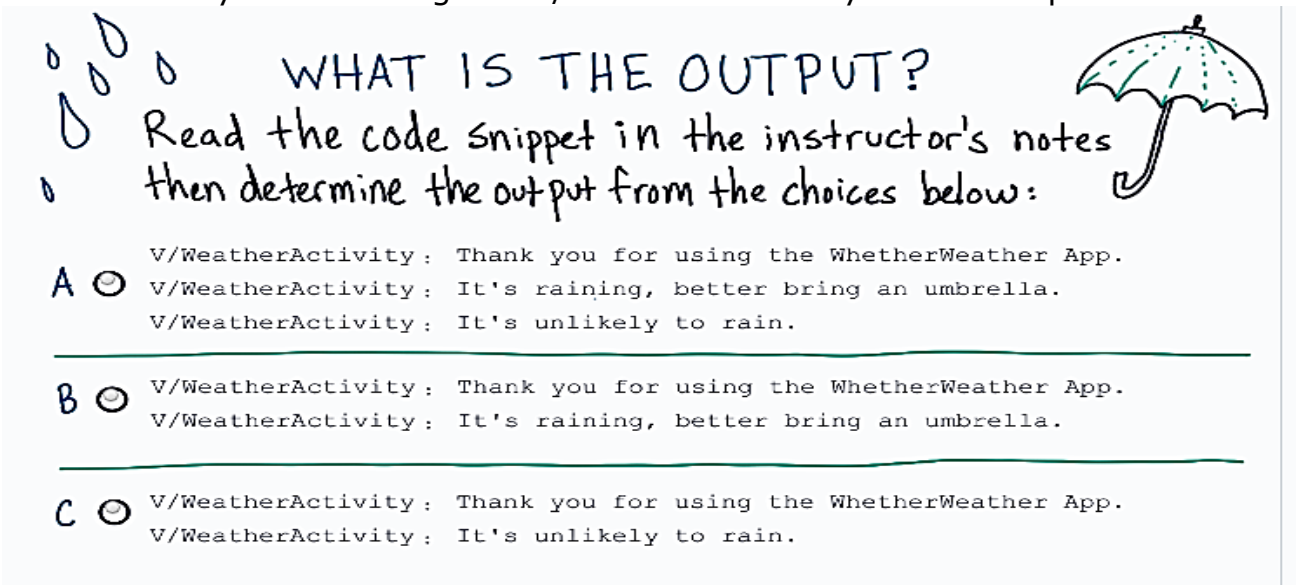





```java
/**
 * This method is called when the order button is clicked.
 */
public void submitOrder(View view) {
    EditText nameField = (EditText) findViewById(R.id.name_field);
    String name = nameField.getText().toString();

    // Figure out if the user wants whipped cream topping
    CheckBox whippedCreamCheckBox = (CheckBox) findViewById(R.id.whipped_cream_checkbox);
    boolean hasWhippedCream = whippedCreamCheckBox.isChecked();

    // Figure out if the user wants chocolate topping
    CheckBox chocolateCheckBox = (CheckBox) findViewById(R.id.chocolate_checkbox);
    boolean hasChocolate = chocolateCheckBox.isChecked();

    int price = calculatePrice();
    String priceMessage = createOrderSummary(name, price, hasWhippedCream, hasChocolate);
    displayMessage(priceMessage);
}
```

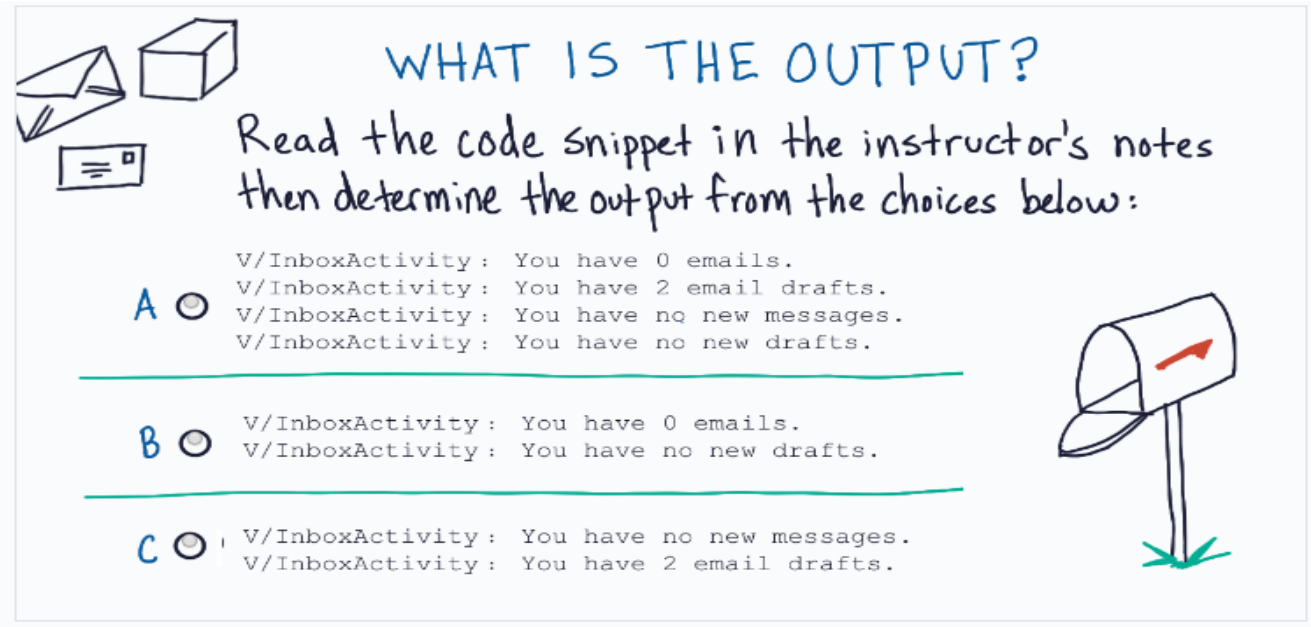
Conditional Code

To test your knowledge on if/else statement try to do this quiz.



Code snippet in WeatherActivity.java — Raw

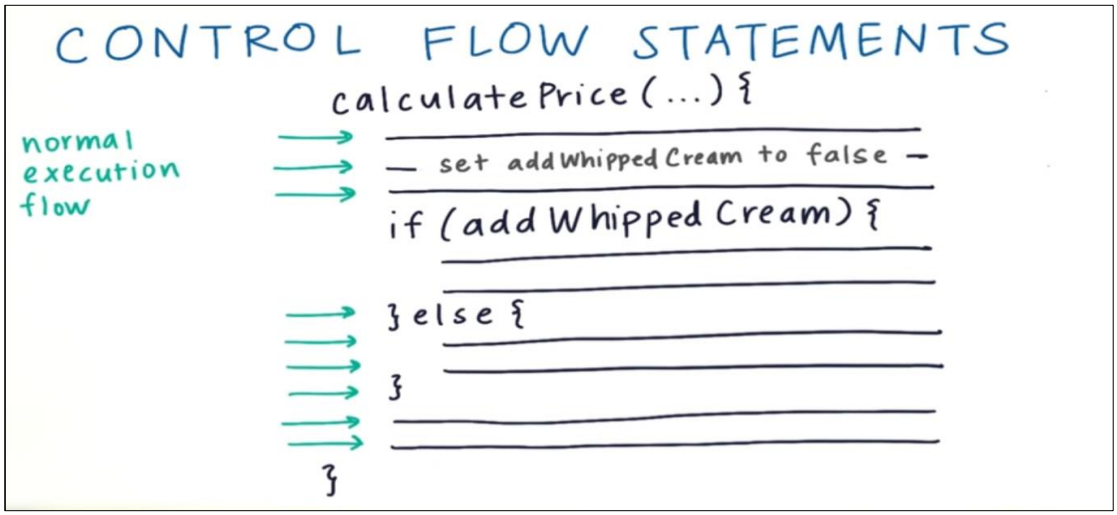
```
boolean isRaining = false;
Log.v("WeatherActivity", "Thank you for using the WhetherWeather App.");
if (isRaining) {
    Log.v("WeatherActivity", "It's raining, better bring an umbrella.");
} else {
    Log.v("WeatherActivity", "It's unlikely to rain.");
}
```

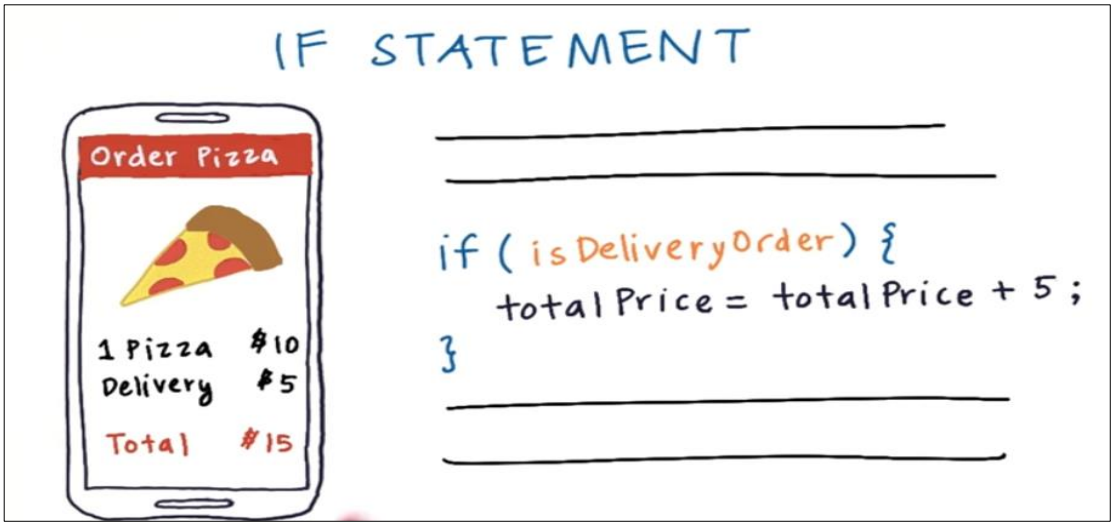


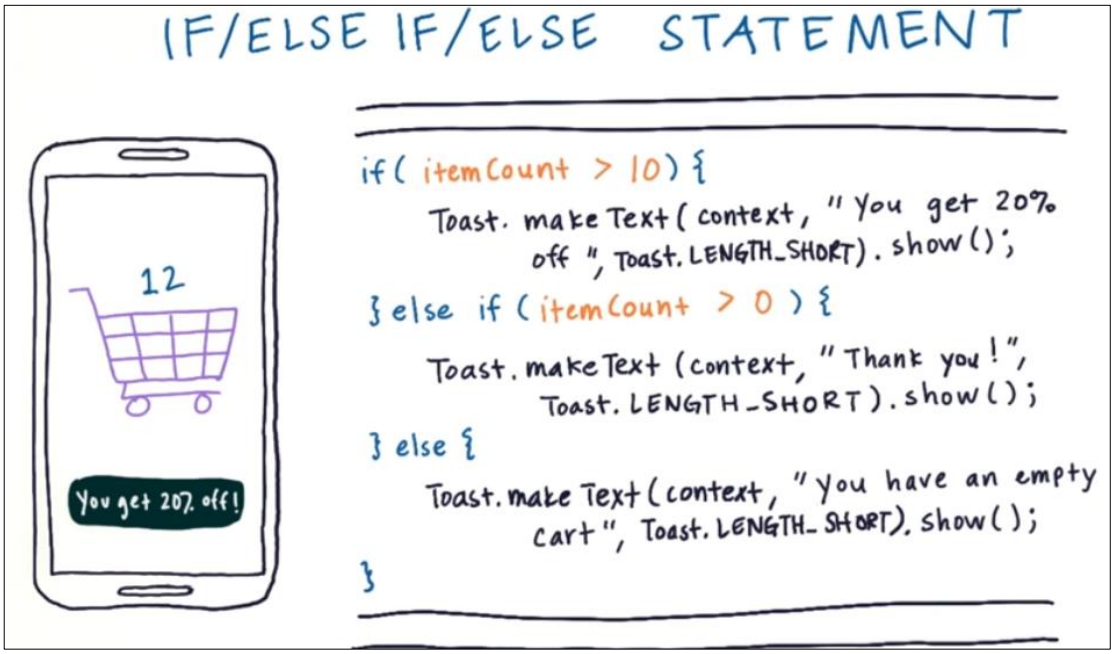
Code snippet in InboxActivity.java — Raw

```
int numberOfEmailsInInbox = 0;
int numberOfDraftEmails = 2;
String emailMessage = "You have " + numberOfEmailsInInbox + " emails. ";
String draftMessage = "You have " + numberOfDraftEmails + " email drafts.";
if (numberOfEmailsInInbox == 0) {
    emailMessage = "You have no new messages. ";
}

if (numberOfDraftEmails == 0) {
    draftMessage = "You have no new drafts.";
}

Log.v("InboxActivity", emailMessage);
Log.v("InboxActivity", draftMessage);
```

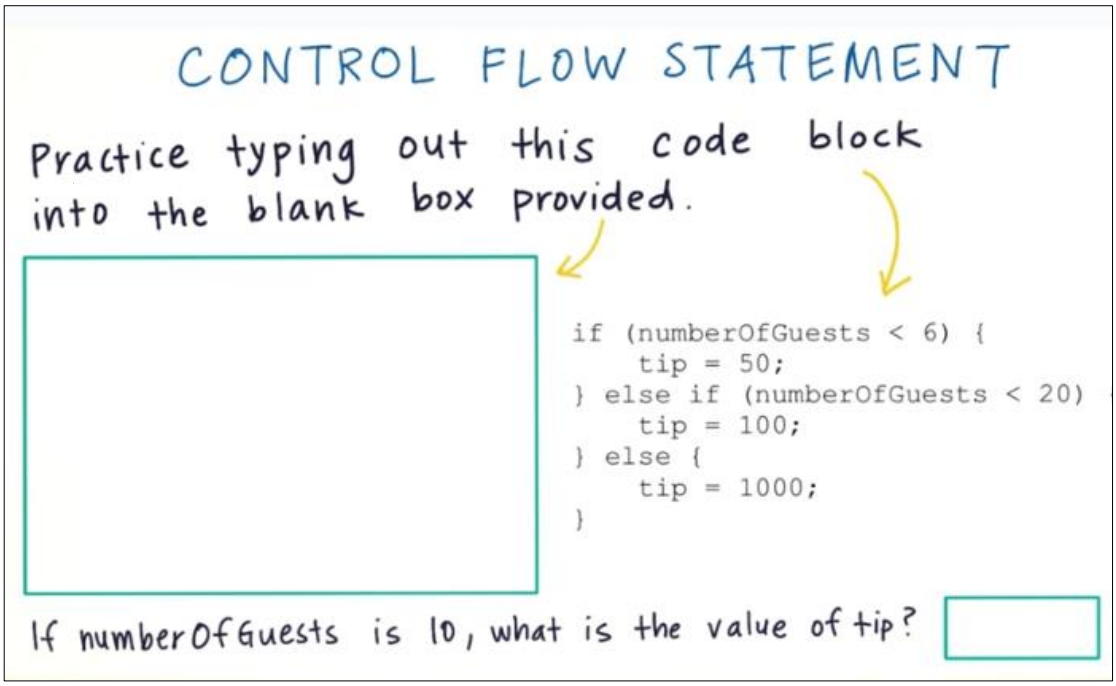
Introduction to If/Else







Syntax of If/Else



Relational Operators

The Equality and Relational Operators in Java

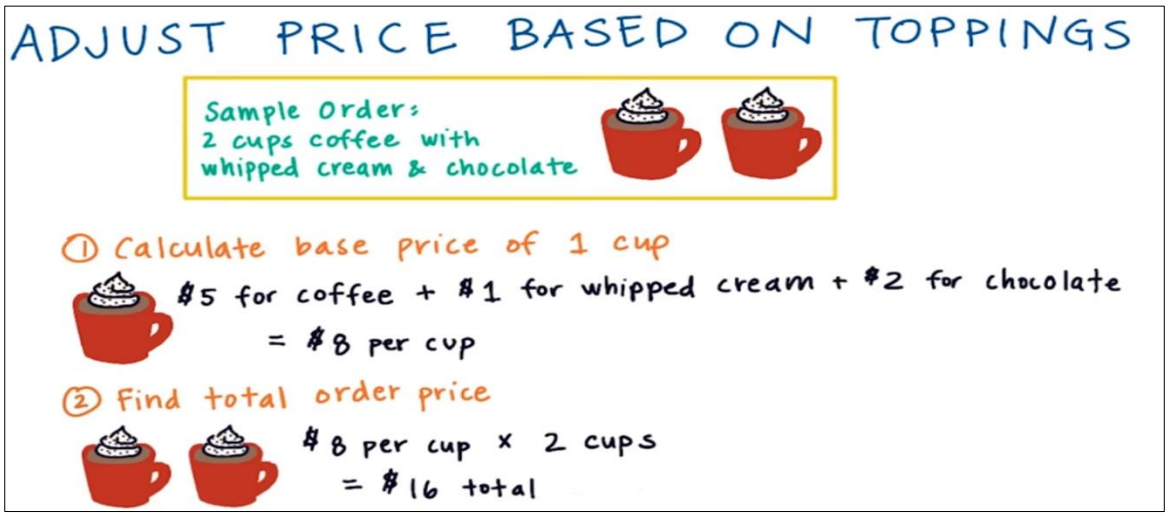
- == Equal to (note that it is not a single = sign because this is the **assignment** operator)
- != Not equal to
- > Greater than
- >= Greater than or equal to
- < Less than
- <= Less than or equal to

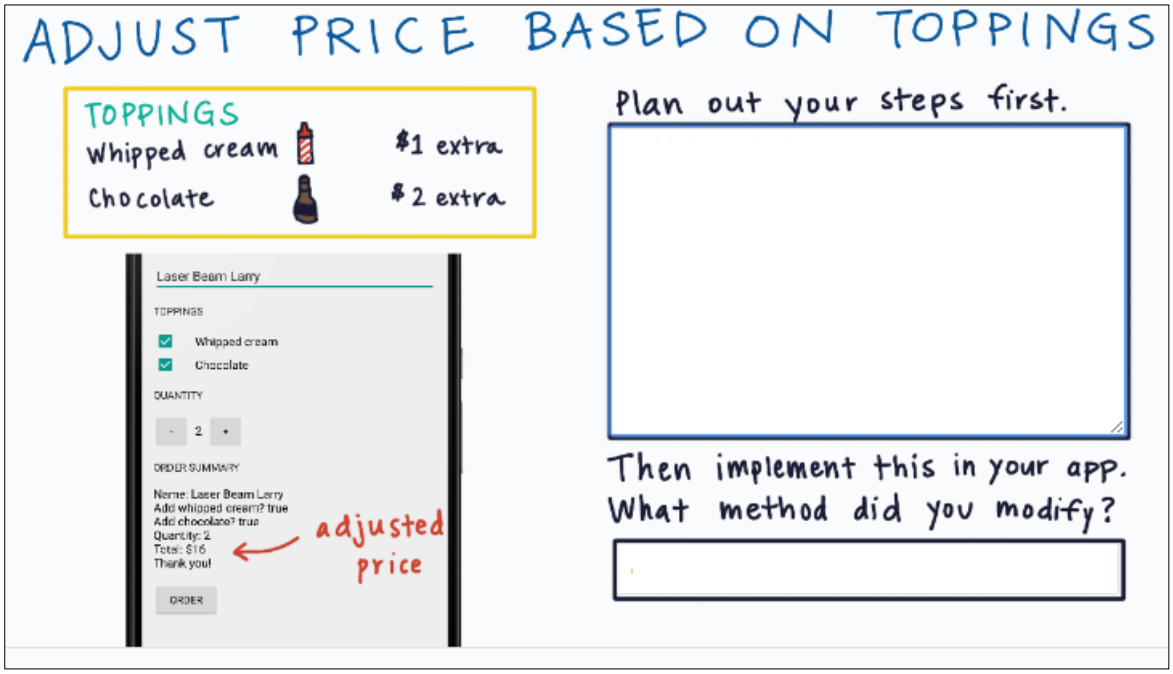
Example

Here's an example of how expressions using relational operators work:

`5 < 10` is an expression meaning 5 is less than 10. This evaluates to the value **true**, because 5 is less than 10.
`5 == 5` evaluates to true.
`5 != 5` evaluates to false. This is because it is false that five is not equal to five.
Much more could be said about relational and equality operators. There is a lot of good information out on the web about these concepts.

Adjust Price with Toppings

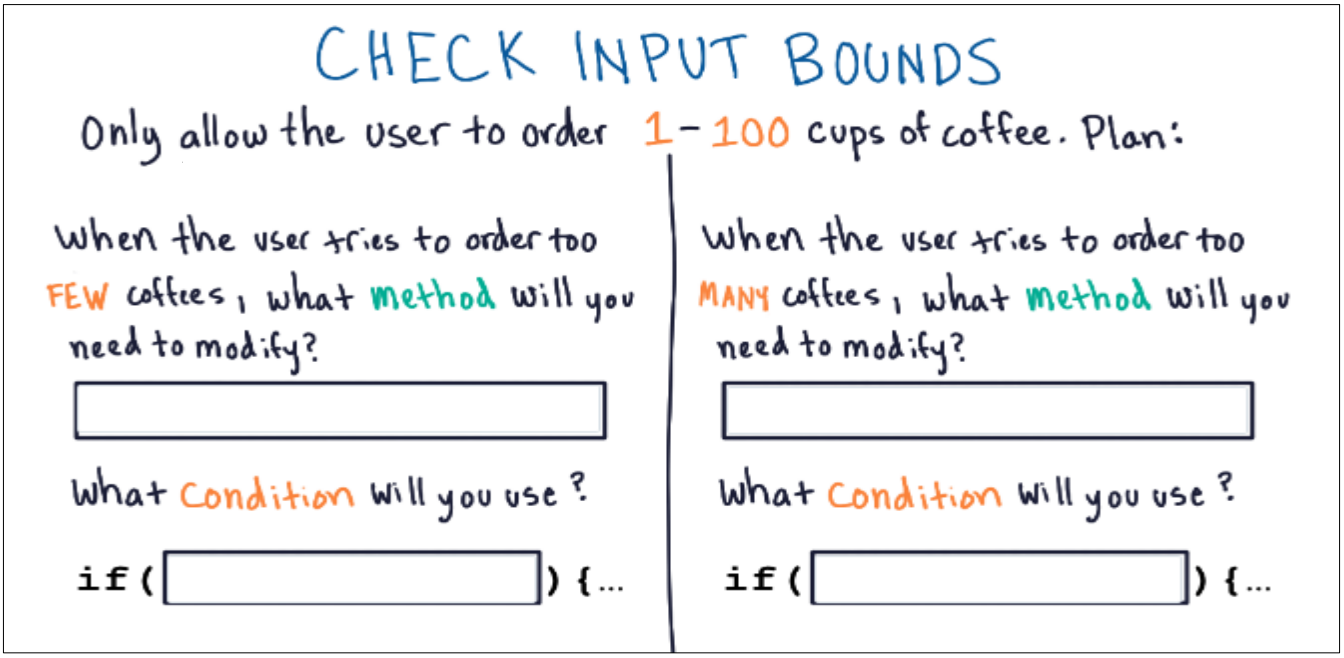




MainActivity.java × activity_main.xml ×

```
64      * Calculates the price of the order.
65      *
66      * @param addWhippedCream is whether or not the user wants whipped cream topping
67      * @param addChocolate is whether or not the user wants chocolate topping
68      * @return total price
69      */
70     private int calculatePrice(boolean addWhippedCream, boolean addChocolate) {
71         // Price of 1 cup of coffee
72         int basePrice = 5;
73
74         // Add $1 if the user wants whipped cream
75         if (addWhippedCream) {
76             basePrice = basePrice + 1;
77         }
78
79         // Add $2 if the user wants chocolate
80         if (addChocolate) {
81             basePrice = basePrice + 2;
82         }
83
84         // Calculate the total order price by multiplying by quantity
85         return quantity * basePrice;
86     }
```

Negative Cups of Coffee



Note that we're looking for the condition that will be true when the number of coffees should *not* change.

Hint: You can put a 'return' keyword inside of an if statement, and that that will end the method early.

```
    /**
     * This method is called when the plus button is clicked.
     */
    public void increment(View view) {
        if (quantity == 100) {
            return;
        }
        quantity = quantity + 1;
        displayQuantity(quantity);
    }

    /**
     * This method is called when the minus button is clicked.
     */
    public void decrement(View view) {
        if (quantity == 1) {
            return;
        }
        quantity = quantity - 1;
        displayQuantity(quantity);
    }
```

Then for adding the Toast message look at this code:

```
/**
 * This method is called when the plus button is clicked.
 */
public void increment(View view) {
    if (quantity == 100) {
        // Show an error message as a toast
        Toast.makeText(this, "You cannot have more than 100 coffees", Toast.LENGTH_SHORT).sho
        // Exit this method early because there's nothing left to do
        return;
    }
    quantity = quantity + 1;
    displayQuantity(quantity);
}

/**
 * This method is called when the minus button is clicked.
 */
public void decrement(View view) {
    if (quantity == 1) {
        // Show an error message as a toast
        Toast.makeText(this, "You cannot have less than 1 coffee", Toast.LENGTH_SHORT).show()
        // Exit this method early because there's nothing left to do
        return;
```

Toasts

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.

The Basics

First, instantiate a `Toast` object with one of the `makeText()` methods. This method takes three parameters: the application `Context`, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with `show()`, as shown in the following example:

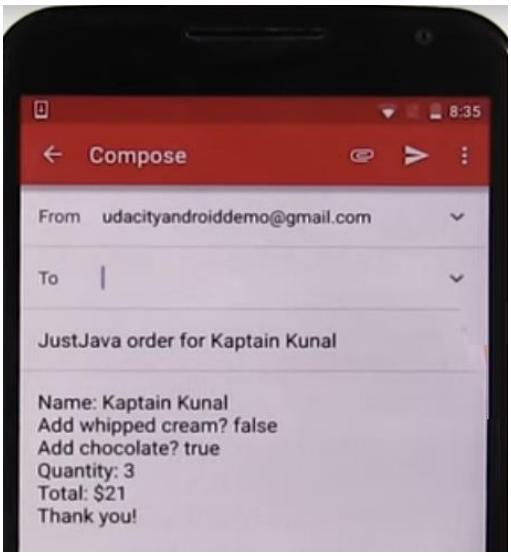
```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

This example demonstrates everything you need for most toast notifications. You should rarely need anything else. You may, however, want to position the toast differently or even use your own layout instead of a simple text message. The following sections describe how you can do these things.

You can also chain your methods and avoid holding on to the Toast object, like this:

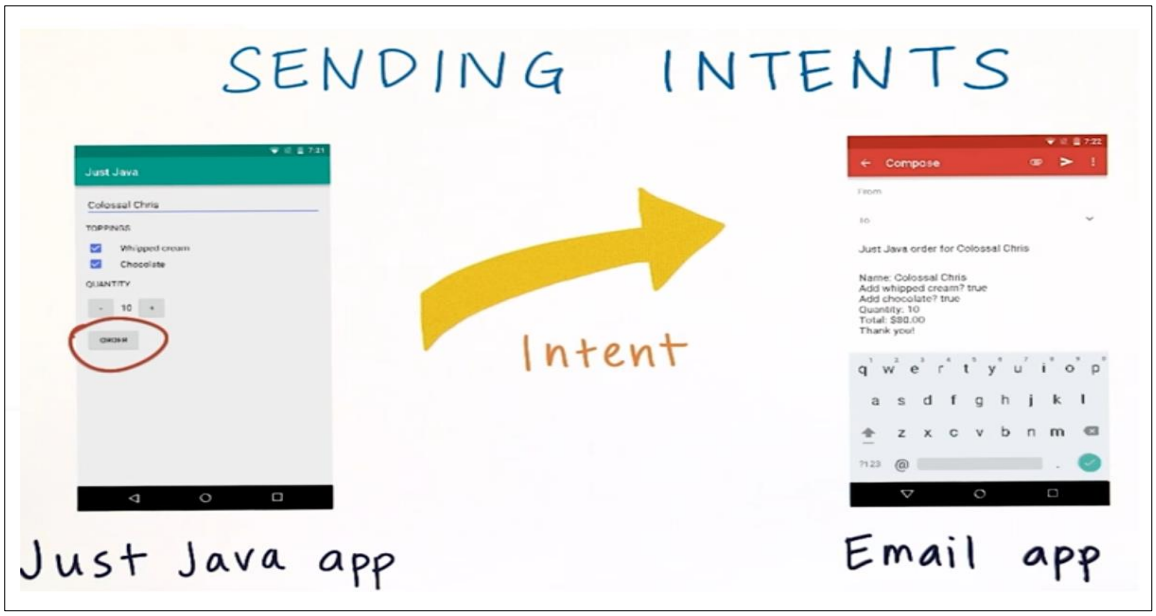
```
Toast.makeText(context, text, duration).show();
```

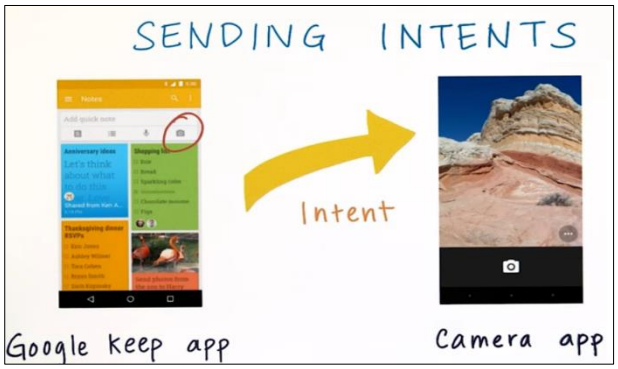
Let Someone Else Do the Hard Work

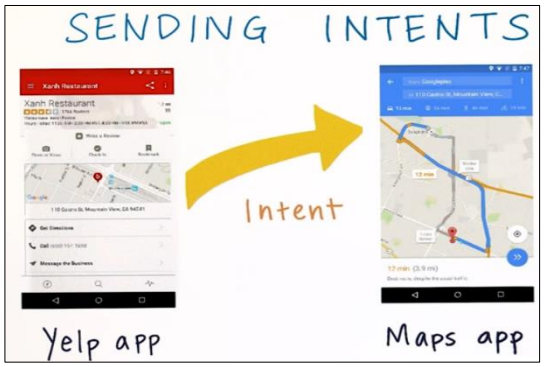


Now that we have the order summary we need to send it off in an email so actually we get our coffee. In order to do this, we must understand a new topic in Android called **intents**.

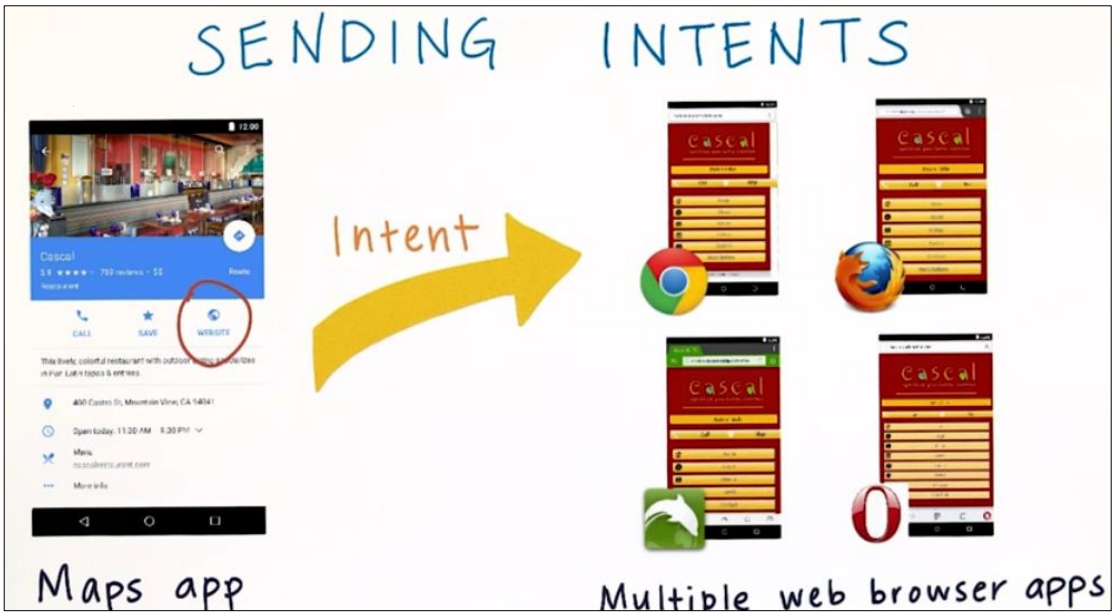
Intent is a message that requests some action to be performed by another app component such as an activity in another app. This is powerful part of Android framework because our App can get functionality that other Apps provide like sending an email opening camera and reviewing a map.

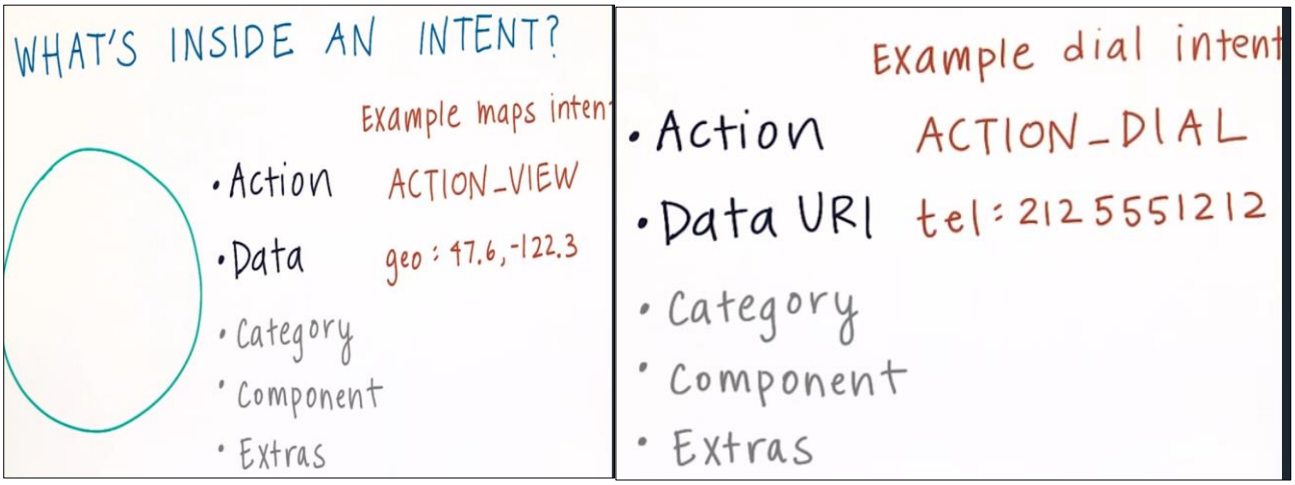






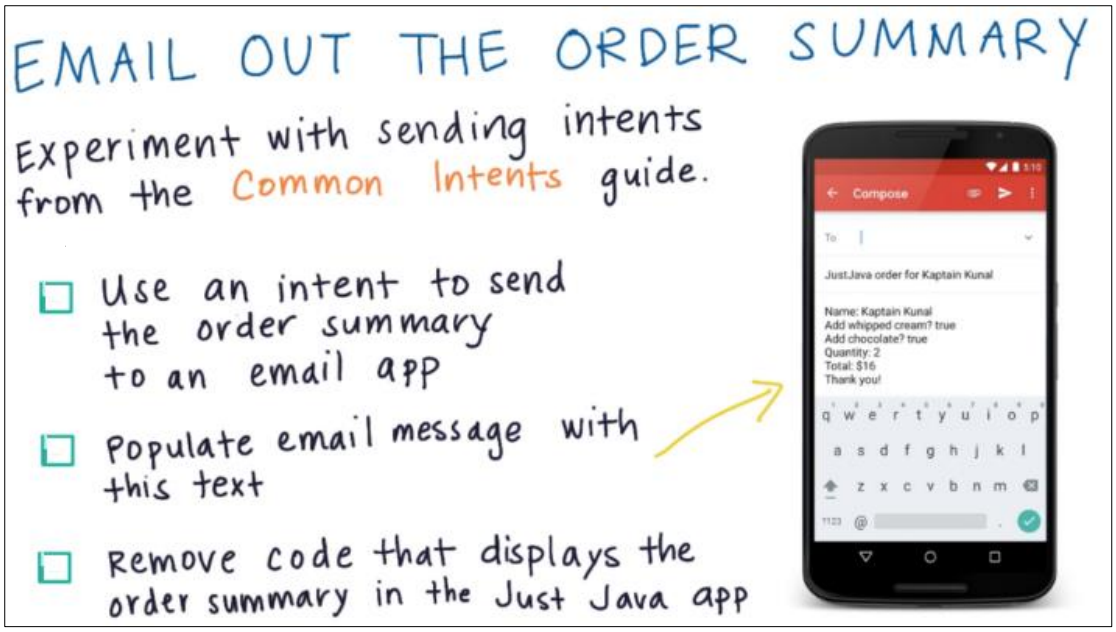
In the example below when the user click on Website and there are multiple browsers Apps in the device, then Android Framework will pop up a message that asks user to select one of these browsers.





Practice with Intents

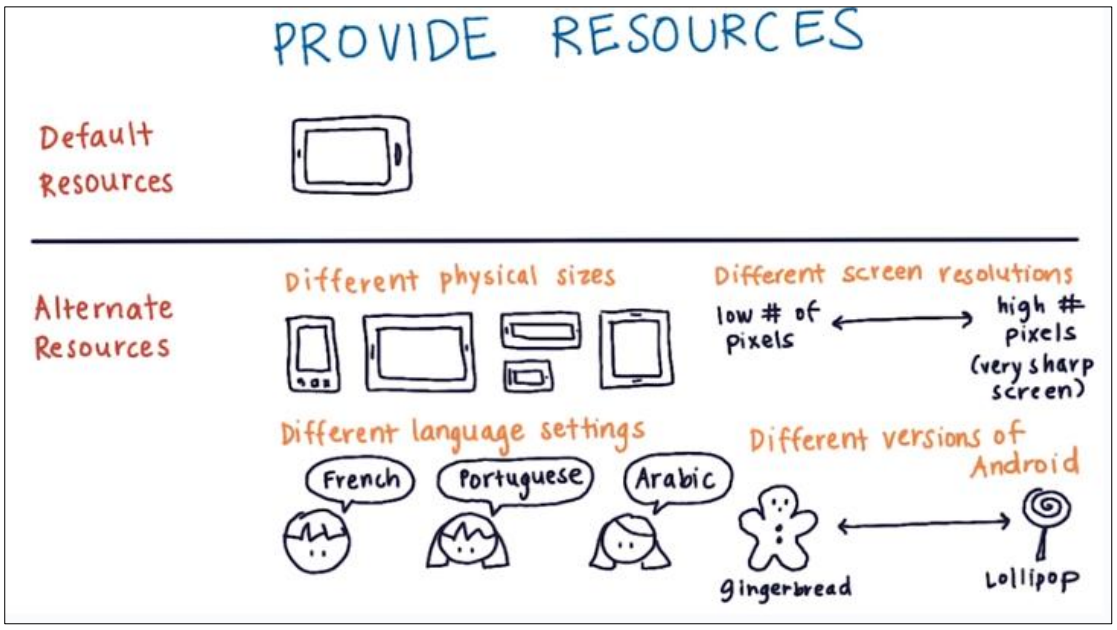
For students using the emulator: This is one of the few pieces of the course that does not work on an emulator, so we suggest you should have an android device to develop with.




```java
public void submitOrder(View view) {
    // Find the user's name
    EditText nameField = (EditText) findViewById(R.id.name_field);
    String name = nameField.getText().toString();

    // Figure out if the user wants whipped cream topping
    CheckBox whippedCreamCheckBox = (CheckBox) findViewById(R.id.whipped_cream_checkbox);
    boolean hasWhippedCream = whippedCreamCheckBox.isChecked();

    // Figure out if the user wants chocolate topping
    CheckBox chocolateCheckBox = (CheckBox) findViewById(R.id.chocolate_checkbox);
    boolean hasChocolate = chocolateCheckBox.isChecked();

    int price = calculatePrice(hasWhippedCream, hasChocolate);
    String priceMessage = createOrderSummary(name, price, hasWhippedCream, hasChocolate);

    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:")); // only email apps should handle this
    intent.putExtra(Intent.EXTRA_SUBJECT, "Just Java order for " + name);
    intent.putExtra(Intent.EXTRA_TEXT, priceMessage);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
```

Localization (OPTIONAL)

When you put your App on Google Play store , it's not easy to expect what type of device that your App will run on, it might be a phone, tablet, older version of Android , or different language than what you using. One of best practice is to local your App which make it localized to a country and language. Now we should work on how to make your App supports different languages. From user side, you can go to your device and choose the language that you prefer. If you choose Arabic all options and menus language will be changed to support Arabic, this also goes to other Apps in your device if the developers chose to



support Arabic. However when you open your Just Java App its unpleasant experience because everything in this App still in English, while other Apps in the device are in Arabic. So how to make our App supports Arabic. You can refer to this link to get full guide on this topic:

Checklist of developer.android.com

1. Identify target languages and locales 2. Design for localization
3. Manage strings for localization 4. Translate UI strings and other resources
5. Test your localized app 6. Prepare for international launch
7. Support international users after launch

Once you've decided on the languages you will support, create the resource subdirectories and string resource files. For example:

```
MyProject/
    res/
        values/
            strings.xml
        values-es/
            strings.xml
        values-fr/
            strings.xml
```

Add the string values for each locale into the appropriate file. At runtime, the Android system uses the appropriate set of string resources based on the locale currently set for the user's device. For example, the following are some different string resource files for different languages.

English (default locale), `/values/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">My Application</string>
    <string name="hello_world">Hello World!</string>
</resources>
```

Spanish, `/values-es/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mi Aplicación</string>
    <string name="hello_world">Hola Mundo!</string>
</resources>
```

French, `/values-fr/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">Mon Application</string>
    <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Use the String Resources

You can reference your string resources in your source code and other XML files using the resource name defined by the `<string>` element's `name` attribute.

In your source code, you can refer to a string resource with the syntax `R.string.<string_name>`. There are a variety of methods that accept a string resource this way.

For example:

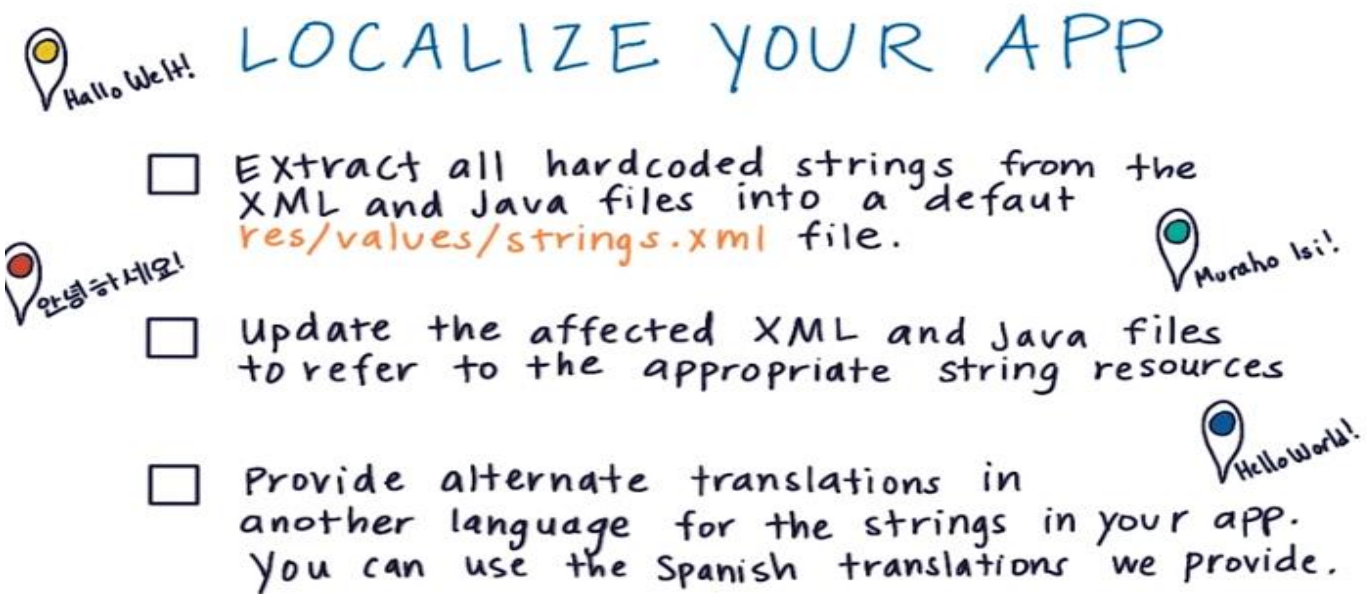
```
// Get a string resource from your app's Resources
String hello = getResources().getString(R.string.hello_world);

// Or supply a string resource to a method that requires a string
TextView textView = new TextView(this);
textView.setText(R.string.hello_world);
```

In other XML files, you can refer to a string resource with the syntax `@string/<string_name>` whenever the XML attribute accepts a string value.

For example:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
```

Now it's time to practice with localization part of your app into Spanish.



Spanish Localization Solution. This would be saved in the res/values-es/strings.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <!-- Title for the application. [CHAR LIMIT=12] -->
    <string name="app_name">Sólo Java</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=20] -->
    <string name="name">Nombre</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=20] -->
    <string name="toppings">Ingredientes</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=20] -->
    <string name="whipped_cream">Crema batida</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=20] -->
    <string name="chocolate">Chocolate</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=20] -->
    <string name="quantity">Cantidad</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=5] -->
    <string name="initial_quantity_value">2</string>

    <!-- Hint text display in the empty field for the user's name
[CHAR LIMIT=20] -->
    <string name="order">Ordenar</string>

<string name="order_summary_email_subject">Sólo java para <xliff:g
id="name" example="Amy">%s</xliff:g></string>
</resources>
```

Styles

A **style** is a collection of properties that specify the look and format for a `View` or window. A style can specify properties such as height, padding, font color, font size, background color, and much more. A style is defined in an XML resource that is separate from the XML that specifies the layout.

Styles in Android share a similar philosophy to cascading stylesheets in web design—they allow you to separate the design from the content.

For example, by using a style, you can take this layout XML:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

And turn it into this:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

All of the attributes related to style have been removed from the layout XML and put into a style definition called CodeFont, which is then applied with the style attribute. You'll see the definition for this style in the following section.

Defining Styles:

To create a set of styles, save an XML file in the res/values/ directory of your project. The name of the XML file is arbitrary, but it must use the .xmlextension and be saved in the res/values/ folder.

The root node of the XML file must be <resources>.

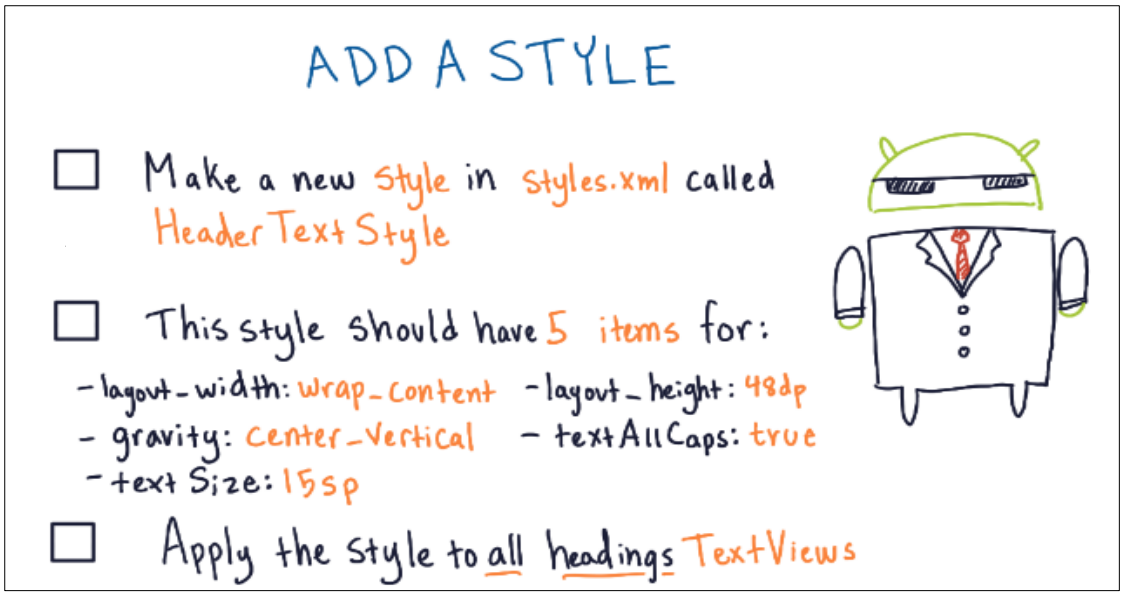
For each style you want to create, add a <style> element to the file with a name that uniquely identifies the style (this attribute is required). Then add an <item> element for each property of that style, with a name that declares the style property and a value to go with it (this attribute is required). The value for the <item> can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property. Here's an example file with a single style:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Each child of the `<resources>` element is converted into an application resource object at compile-time, which can be referenced by the value in the `<style>` element's `name` attribute. This example style can be referenced from an XML layout as `@style/CodeFont` (as demonstrated in the introduction above).

The `parent` attribute in the `<style>` element is optional and specifies the resource ID of another style from which this style should inherit properties. You can then override the inherited style properties if you want to.

Remember, a style that you want to use as an Activity or application theme is defined in XML exactly the same as a style for a View. A style such as the one defined above can be applied as a style for a single View or as a theme for an entire Activity or application. How to apply a style for a single View or as an application theme is discussed later.



Themes

A **theme** is a **style** applied to an entire Activity or application, rather than an individual View (as in the example above). When a style is applied as a theme, every View in the Activity or application will apply each style property that it supports. For example, you can apply the same CodeFont style as a theme for an Activity and then all text inside that Activity will have green monospace font.

To set a theme for all the activities of your application, open the **AndroidManifest.xml** file and edit the **<application>** tag to include the **android:theme** attribute with the style name. For example –

```
<application android:theme="@style/CustomFontStyle">
```

But if you want a theme applied to just one Activity in your application, then add the android:theme attribute to the <activity> tag only. For example –

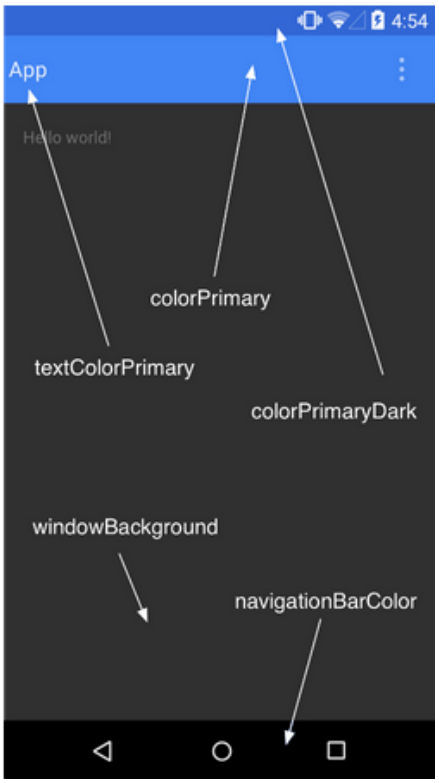
```
<activity android:theme="@style/CustomFontStyle">
```

There are number of default themes defined by Android which you can use directly or inherit them using **parent** attribute as follows –

```
<style name="CustomTheme" parent="android:Theme.Light">
   ...
</style>
```

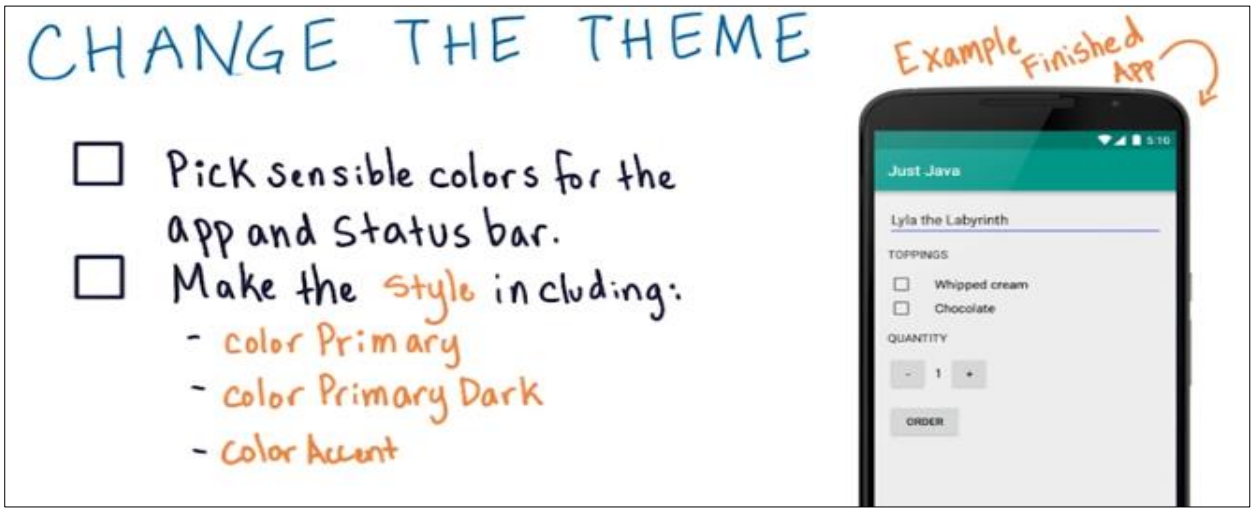
Styling the colour palette

The layout design can implementable based on them based colours, for example as following design is designed based on them color(blue).



Above layout has designed based on style.xml file,Which has placed at **res/values/**

```
<resource>
   <style name="AppTheme"
parent="android:Theme.Material">
      <item name
="android:color/primary">@color/primary</item>
      <item name
="android:color/primaryDark">@color/primary_dark</item>
      <item name
="android:colorAccent/primary">@color/accent</item>
   </style>
<resource>
```



Congratulations for finishing 1st course of Mobile Programming, there are many other things you can do with Android and I will post this article at your **IT412MP group.**

Intents and Activities

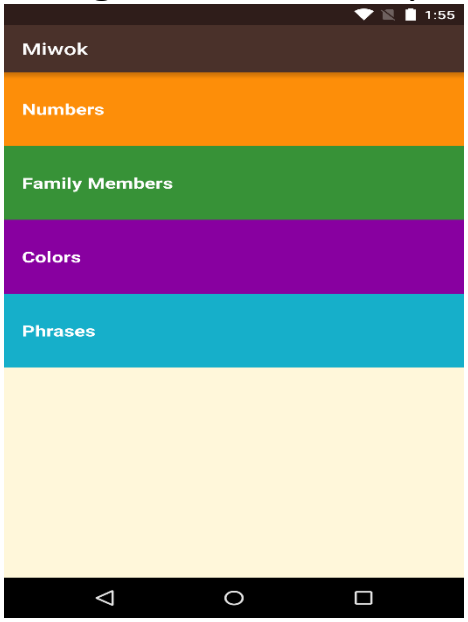
Import an Existing Project

We've created an initial version of the Miwok app for you to start off with, but don't worry--there's still plenty left for you to do. When a professional Android developer is joining a new project, it's very common for the team to already have an existing app that they're working on. It's good practice to learn how to read an existing codebase because, oftentimes, you won't be creating a brand new app from scratch.

Follow these instructions to import the Miwok project into Android Studio on your computer.

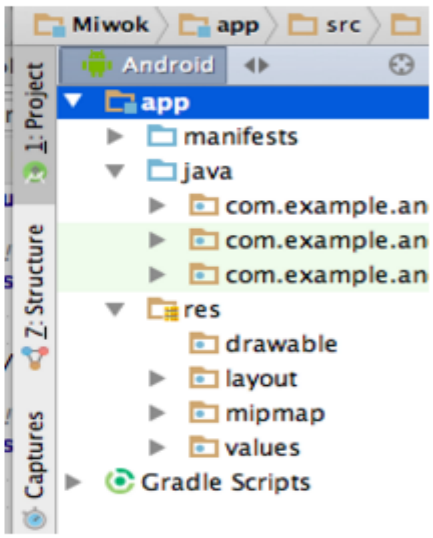
1. Go to **Code1**.
2. Click on the "Download zip" button to download the app code.
3. Unzip the downloaded file on your computer so that you have a "Miwok" folder.
4. Open Android Studio.
5. Choose File > Open and select the "Miwok" folder. It may take some time for the project to be imported. If you have any issues, check the Troubleshooting document.

6. Once that app has successfully imported, run the app on your Android device (phone, tablet, or emulator). It should look like this screenshot. Nothing happens when you click on any of the categories. That's expected.



Once the starter code is running, nothing will happen when you click on any of the categories

7. Open the Project tab in Android Studio so you can see the file directory.

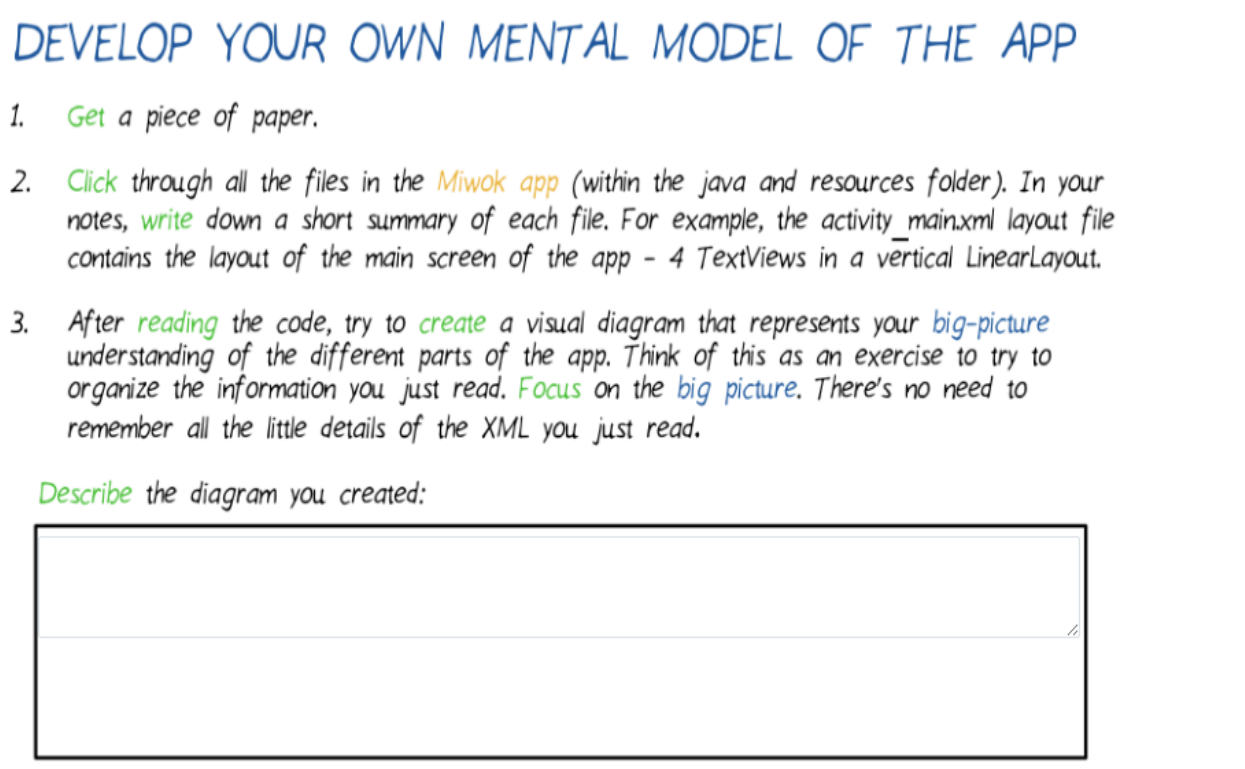


Click on Project tab to see file directory

8. When you're ready, move onto the quiz. Let's take a closer look at the codebase for the app (where codebase means all the files in the app). This will help us understand why the app looks and behaves the way it currently does.

At this point, it's totally normal if you feel nervous about looking at a new codebase because everything looks so foreign. Now is a good time to practice this skill with some guidance from us.

Quiz:

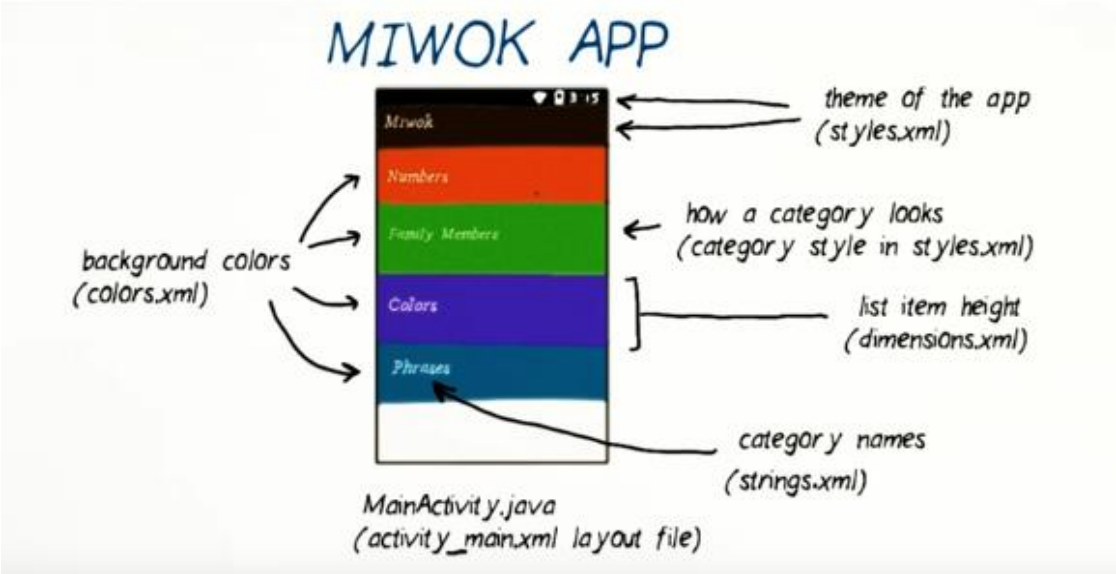


Note:

Ensure that you have the latest extras (Android Studio library, Google Repository, etc) from the Android SDK Manager installed.

Ensure that you have all the required SDKs installed from the Android SDK Manager. For this course you will need SDK 15 through SDK 23.

Sol:



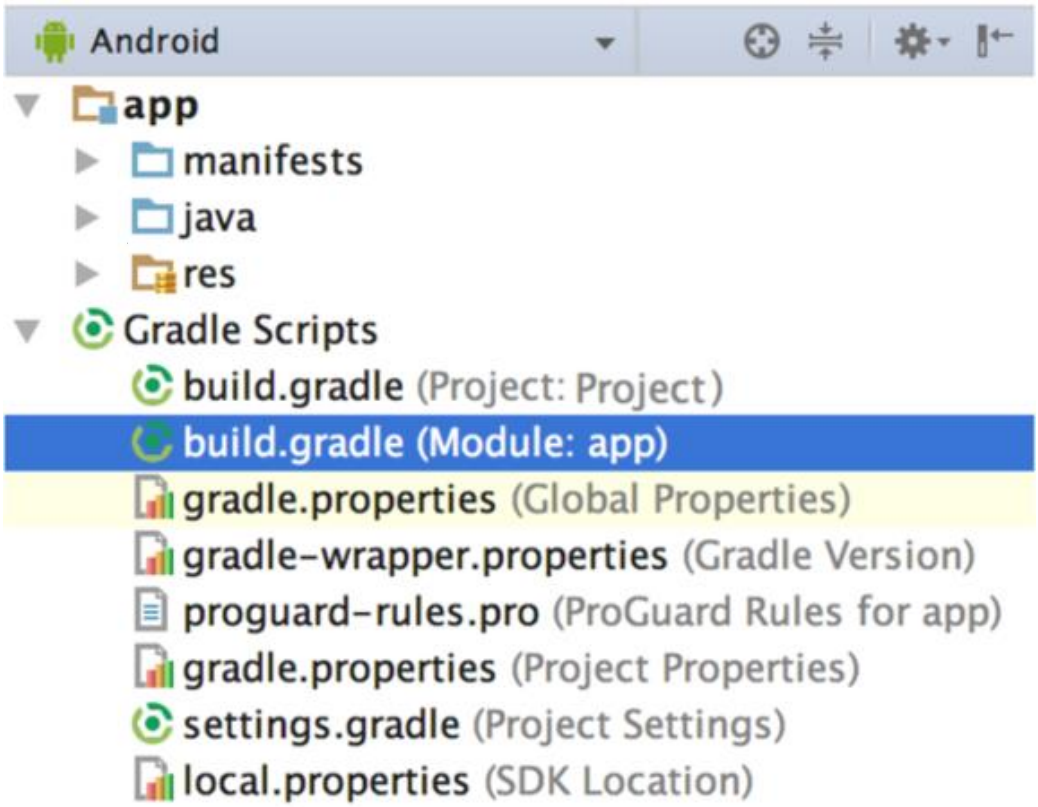
Check build.gradle File

When importing someone else's code into Android Studio, you usually want to use the same package versions and target SDK settings that the original code was designed to use, however keeping your own projects up-to-date is always a good practice.

Because Android keeps releasing new SDK package versions, whenever you open up or import an old project in Android Studio, you might get a warning saying that the plugin is too old:

Luckily Android Studio offers to fix this for you by updating the *build.gradle* file automatically. So clicking "Fix plugin and sync" should do the trick.

However, if - for any reason - you wanted to update the build.gradle yourself, you can find it by browsing to Gradle Scripts and then double click on the build.gradle (Module: app) file:



Make sure that they include the most up-to-date versions of:

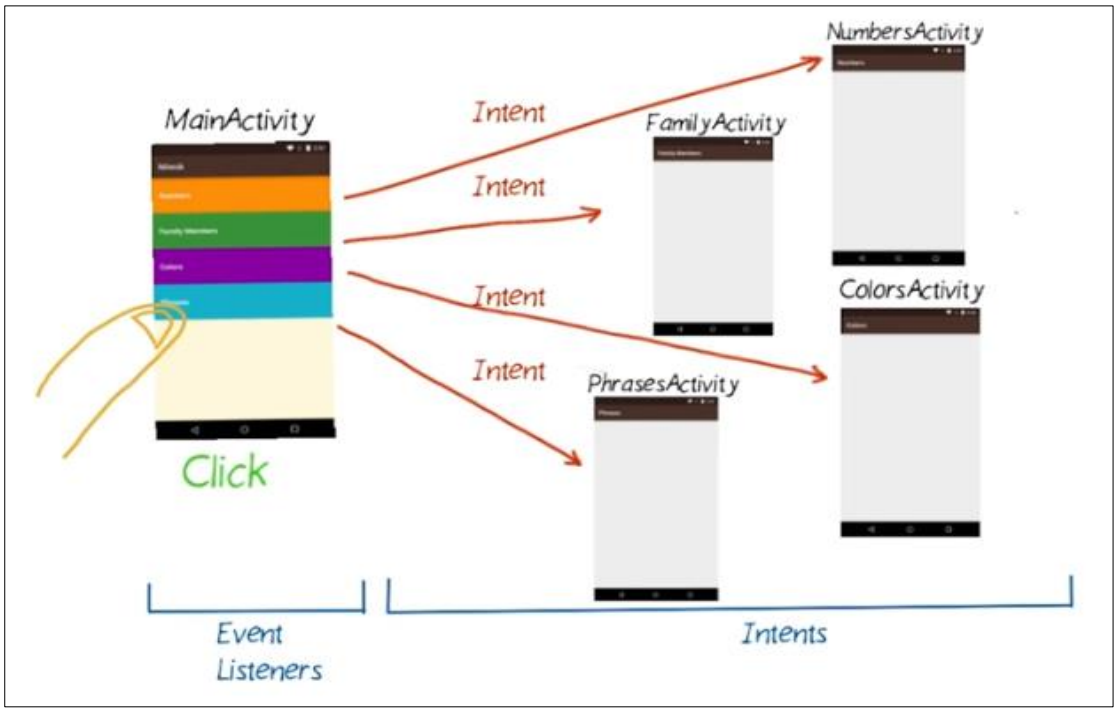
- compileSdkVersion
- buildToolsVersion
- targetSdkVersion

Also, in the same file check that all your dependencies are up-to-date as well, for example when the Constraint Layout beta version was released it would complain about using an alpha version since it's now obsolete:

Usually Android Studio would usually highlight any out-of-date packages like for example:

Read MainActivity.java file

In this lesson we are going to learn how to create new **activities** and navigate to them using **Intents** , that's allow us to create **multiple screens App.** This involves revisiing many concepts like intents and learn about event listeners with a new concept called **interfaces**.



First we should dive deeper into **MainActivity.java** file. Read through the file as best as you can, after that complete the following **matching exercise**

Read MainActivity.java File

Match the text description with the appropriate block of code by putting the correct letter A - F in the blanks provided.

__ Class declaration

__ Package name

__ Method override

__ Import statement

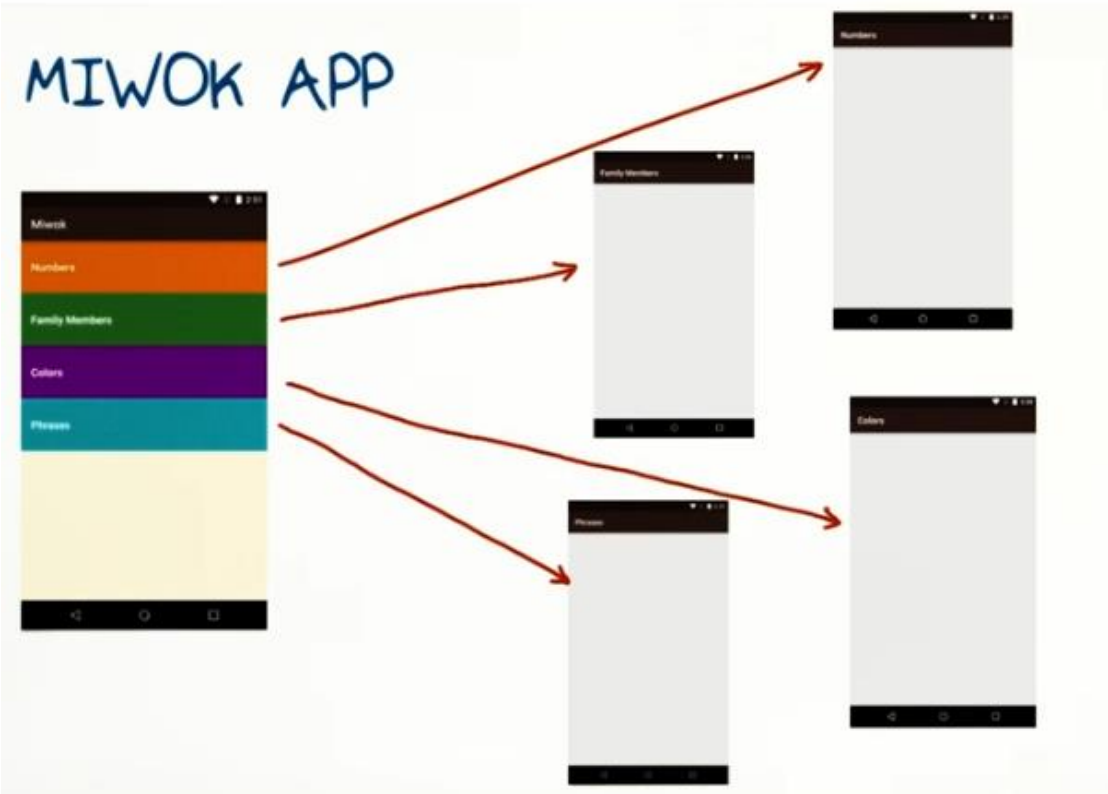
__ Set content view to be an XML layout resource

A
B
C
D
E

```
1  package com.example.android.miwok;
2
3  import android.os.Bundle;
4  import android.support.v7.app.AppCompatActivity;
5
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState)
10         super.onCreate(savedInstanceState);
11
12         // Set the content of the activity to use the
13         setContentView(R.layout.activity_main);
14     }
15 }
```

Create New Activities

Last time we used Intents to send our data to **email App,** but this time we use Intents to open other activities that are created inside Miwok App. We should have **five activities** in our App. Main activity and four activities for the four categories of words.



So let's work on creating these four activities in our App.

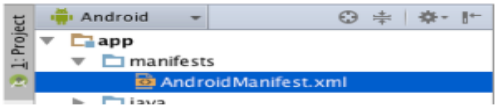
1- Right click on the package in the project directory (**see next page**)
2- When the wizard pops up, give the activity a name, in this case we will use **NumbersActivity,** and accept the defaults and click Finish.

Now it's your turn to create the other activities for the remaining categories, phrases, colors and family members. Try this quiz:

CREATE NEW ACTIVITIES

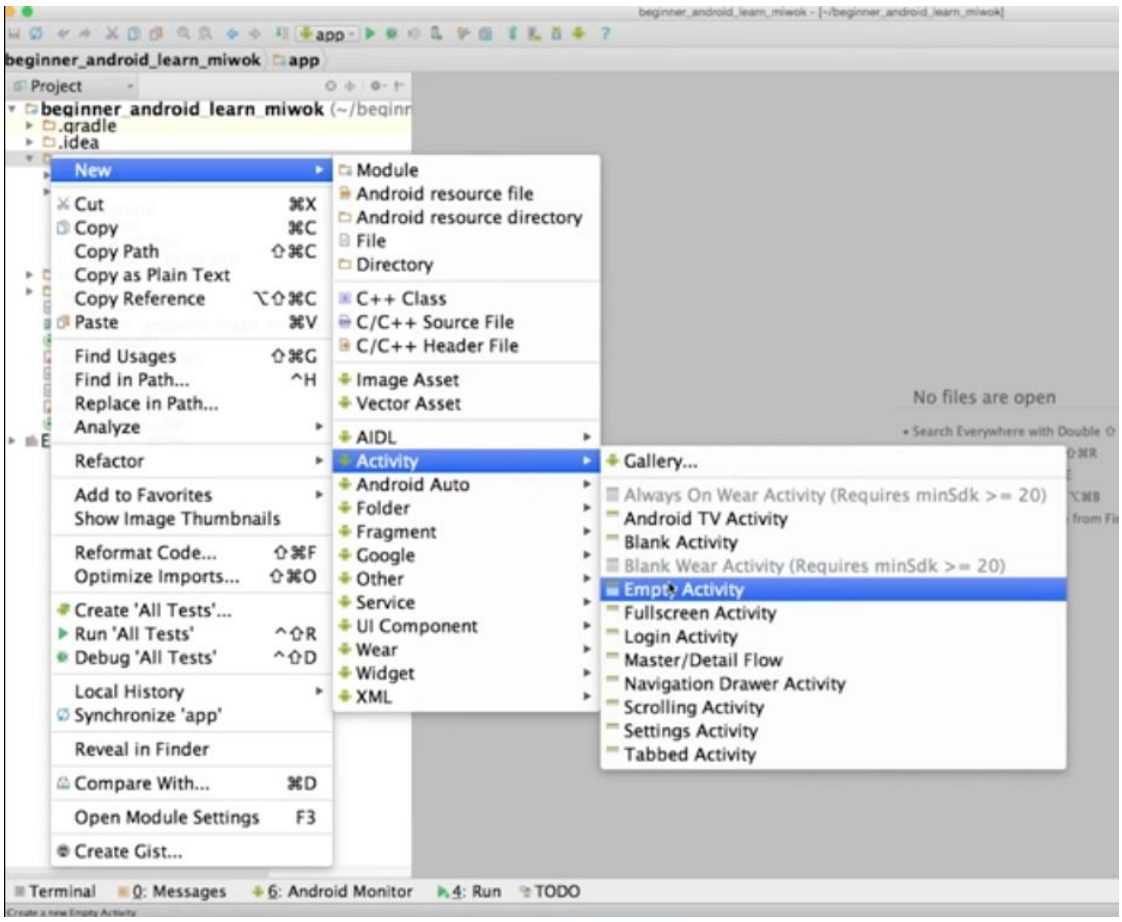
1. ___ Create 4 new activities in the Miwok app:
PhrasesActivity, NumbersActivity, FamilyActivity, ColorsActivity

2a. Navigate to the AndroidManifest.xml file under the app>manifests folder in the project directory:

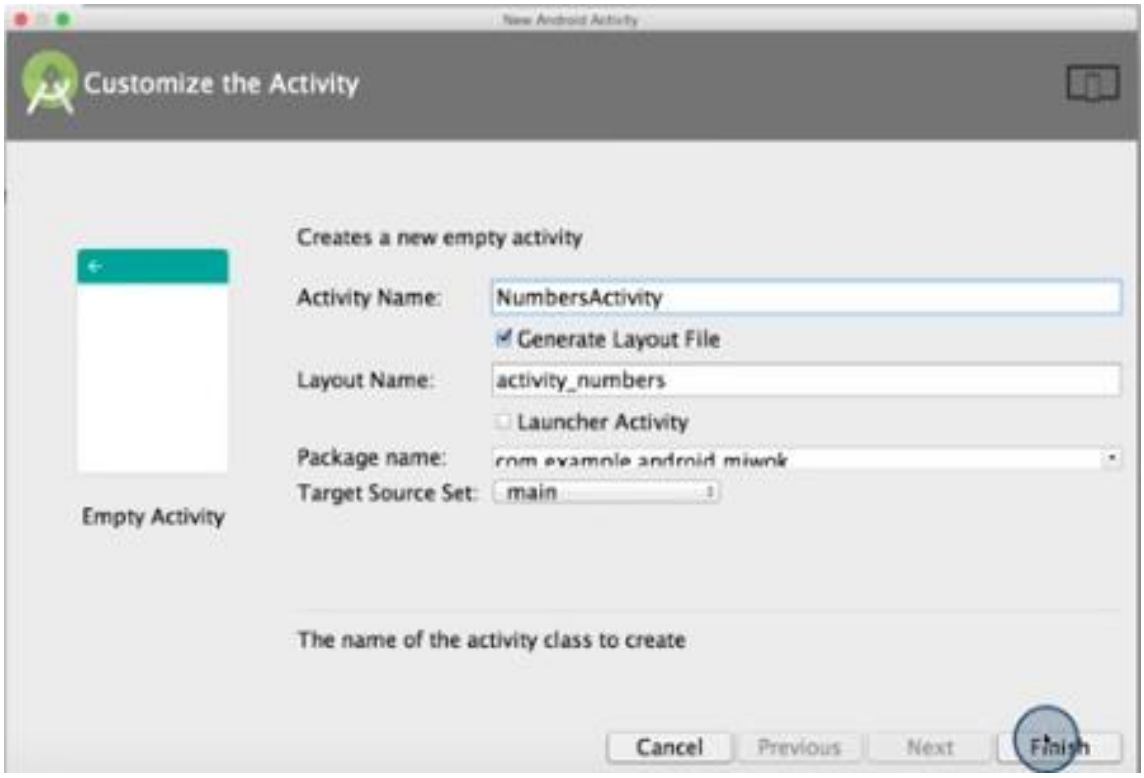


2b. How does the AndroidManifest.xml file change as you add more activities to the app?

Step 1


beginner_android_learn_miwok
app
Project
beginner_android_learn_miwok (~/beginr
.gradle
.idea
New
Cut ⌘X
Copy ⌘C
Copy Path ⇧⌘C
Copy as Plain Text
Copy Reference ⌥⇧⌘C
Paste ⌘V
Find Usages ⇧⌘G
Find in Path... ^H
Replace in Path...
Analyze
Refactor
Add to Favorites
Show Image Thumbnails
Reformat Code... ⇧⌘F
Optimize Imports... ⇧⌘O
Create 'All Tests'...
Run 'All Tests' ^⇧R
Debug 'All Tests' ^⇧D
Local History
Synchronize 'app'
Reveal in Finder
Compare With... ⌘D
Open Module Settings F3
Create Gist...
Module
Android resource file
Android resource directory
File
Directory
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
AIDL
Activity
Android Auto
Folder
Fragment
Google
Other
Service
UI Component
Wear
Widget
XML
Gallery...
Always On Wear Activity (Requires minSdk >= 20)
Android TV Activity
Blank Activity
Blank Wear Activity (Requires minSdk >= 20)
Empty Activity
Fullscreen Activity
Login Activity
Master/Detail Flow
Navigation Drawer Activity
Scrolling Activity
Settings Activity
Tabbed Activity
No files are open
Terminal
0: Messages
6: Android Monitor
4: Run
TODO

Step 2


Customize the Activity
Creates a new empty activity
Activity Name: NumbersActivity
Generate Layout File
Layout Name: activity_numbers
Launcher Activity
Package name:
Target Source Set: main
Empty Activity
The name of the activity class to create
Cancel
Previous
Next
Finish

App Manifest

Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

Among other things, the manifest file does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application, which include the activities, services, broadcast receivers, and content providers that compose the application. It also names the classes that implement each of the components and publishes their capabilities, such as the `Intent` messages that they can handle. These declarations inform the Android system of the components and the conditions in which they can be launched.
- It determines the processes that host the application components.
- It declares the permissions that the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.
- It lists the `Instrumentation` classes that provide profiling and other information as the application runs. These declarations are present in the manifest only while the application is being developed and are removed before the application is published.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.miwok" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Miwok"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".NumbersActivity" >
        </activity>
        <activity android:name=".FamilyActivity" >
        </activity>
        <activity android:name=".ColorsActivity" >
        </activity>
        <activity android:name=".PhrasesActivity" >
        </activity>
    </application>

</manifest>
```

Use an Intent to Open another Activity

By taking a look on the code snippets below, you should be able to figure out on how to use Intent to open **NumberActivity**.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
public void openOne (View view) {
    Intent intent = new Intent(this,ActivityOneAppOne.class);
    startActivity(intent);
}
```

Now it's your turn to try this exercise:

USE INTENT TO OPEN ANOTHER ACTIVITY

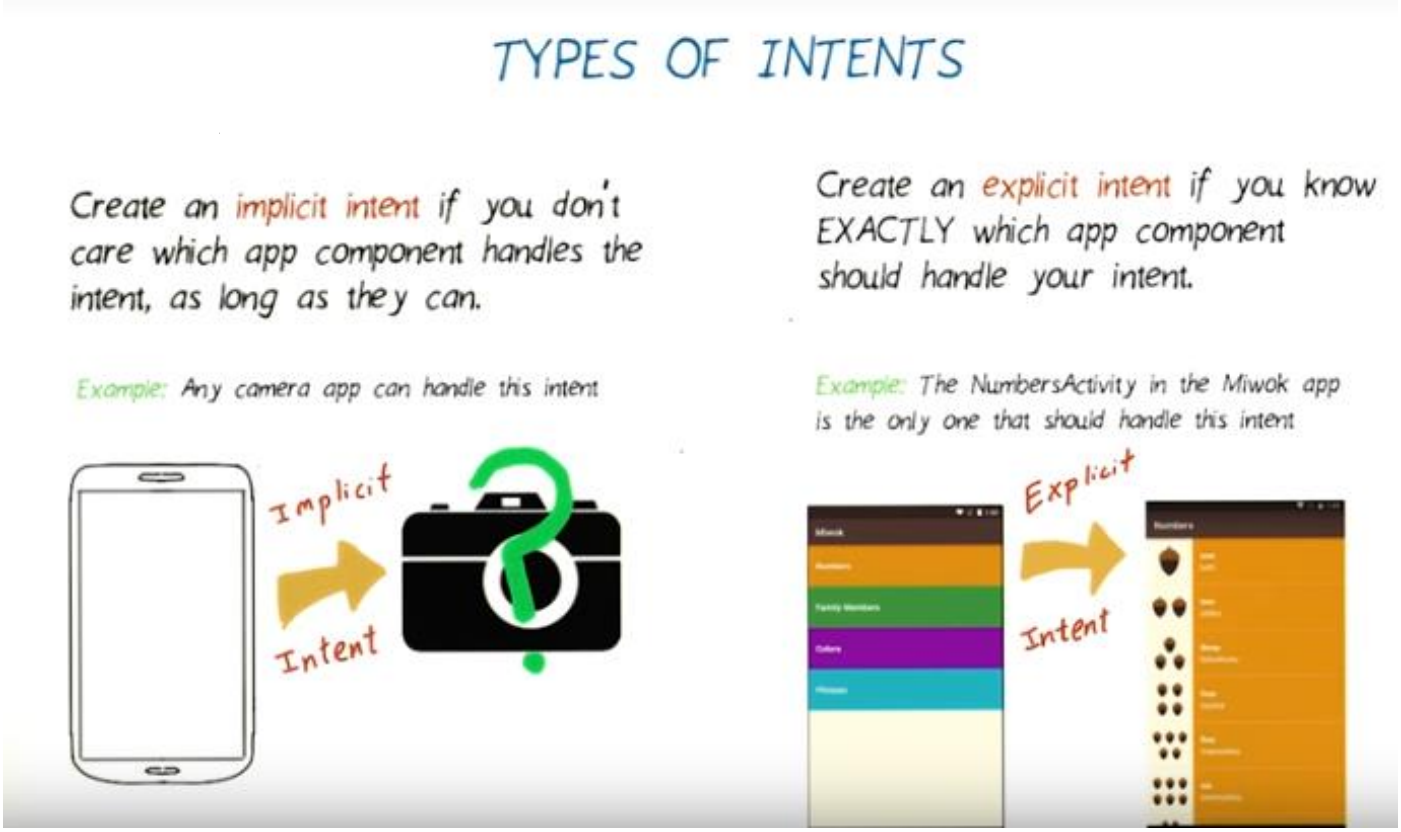
- [ ] Modify the app so that clicking on the "Numbers" category opens up the NumbersActivity

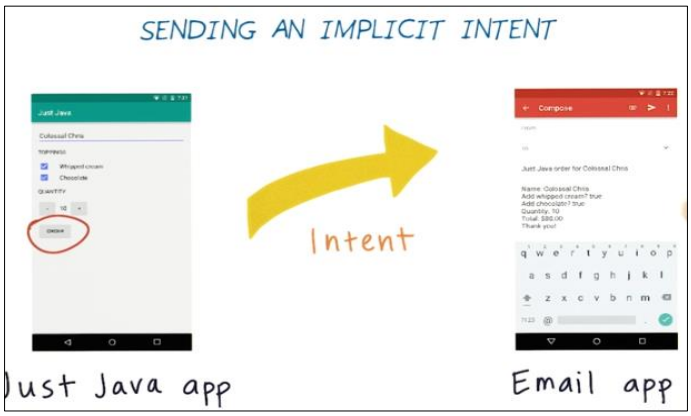
Don't worry about the other 3 categories for now.

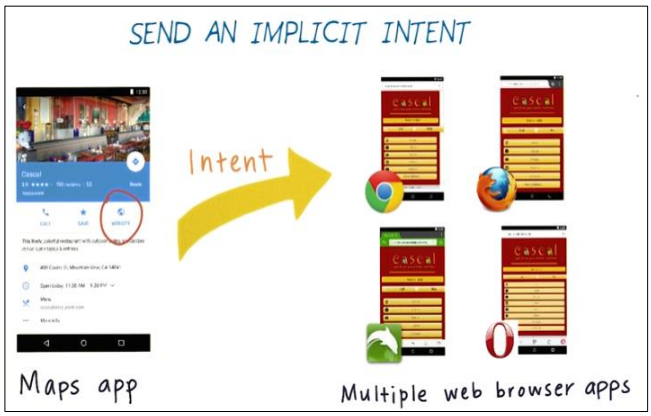

```
29          // Set the content of the activity to use the activity_main.xml layout file
30          setContentView(R.layout.activity_main);
31      }
32
33      public void openNumbersList(View view) {
34          Intent i = new Intent(this, NumbersActivity.class);
35          startActivity(i);
36      }
37  }
```

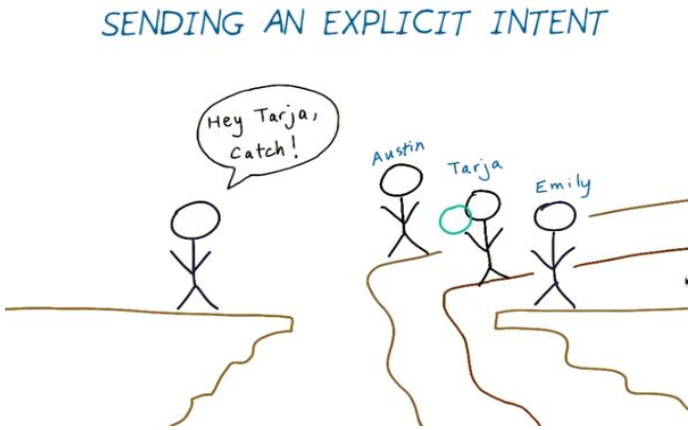
```
<TextView
    android:id="@+id/numbers"
    style="@style/CategoryStyle"
    android:background="@color/category_numbers"
    android:text="@string/category_numbers"
    android:onClick="openNumbersList" />
```

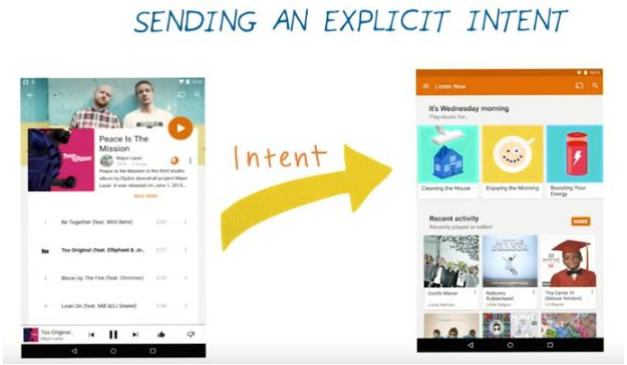
Implicit vs. Explicit Intents











COMPARE & CONTRAST

Implicit Intent

```
// Create the text message with a string
Intent intent = new Intent(Intent.ACTION_SENDTO);
sendIntent.setData(Uri.parse("mailto:"));
sendIntent.putExtra(Intent.EXTRA_SUBJECT, "Just Java order for " + name);
sendIntent.putExtra(Intent.EXTRA_TEXT, priceMessage);

// Verify that the intent will resolve properly
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Explicit Intent

```
// Executed in an Activity, so 'this' is the Context
Intent intent = new Intent(this, NumbersActivity.class);
startActivity(intent);
```

For the following scenarios, determine whether we should use an implicit or explicit intent?

IMPLICIT vs. EXPLICIT INTENT

Case #1:

Popular group texting apps allow users to open video links that they receive in chat messages. Should opening a video player use an implicit or explicit intent?

_ Implicit Intent _ Explicit Intent

Case #2:

In a food order app, a user selects the dishes she wants to order and clicks the "Submit Order" button. The app moves from the Order Activity to the Order Summary Activity. Should moving to the Order Summary Activity use an implicit or explicit intent?

_ Implicit Intent _ Explicit Intent

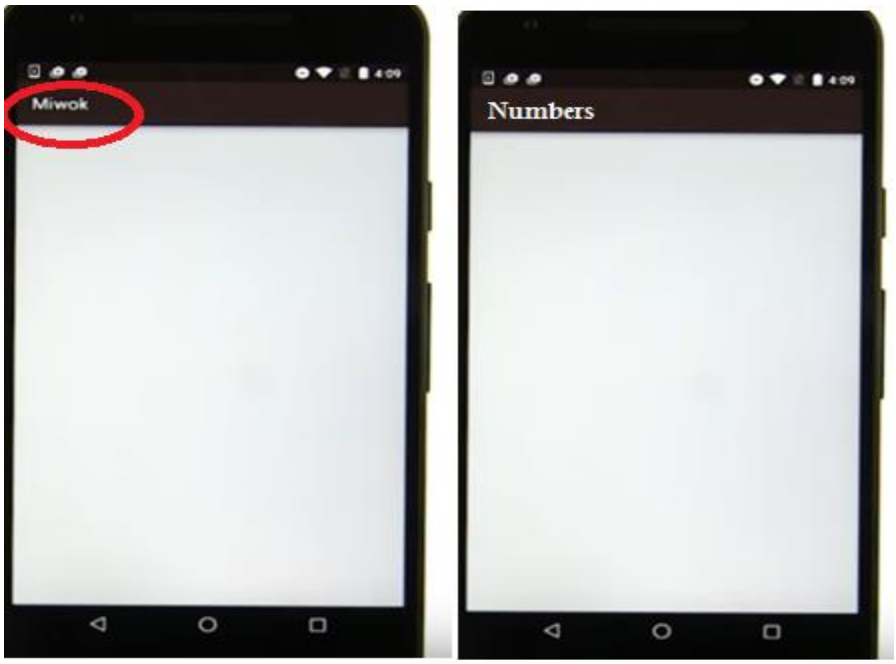
Case #3:

Ride sharing apps allow users to contact their drivers via text messaging. Should accessing a text messaging app on the phone use an implicit or explicit intent?

_ Implicit Intent _ Explicit Intent

Modifying the Activity Name

Now everything is awesome except the activity title which still **Miwok**, and we want to change it to **numbers**. To change the title we should modify **manifest.xml** file. Then we'll add an extra attribute called **android label** and the value will be the text that we want to show in the app bar for this activity which is **numbers** and this name is already exist in strings.xml file.



How to Create an Event Listener

1. Define the event listener (and the custom behavior for when the event happens)

2. Create a new object instance of the event listener (using the constructor)

3. Attach the listener to the view

SETUP AN EVENT LISTENER

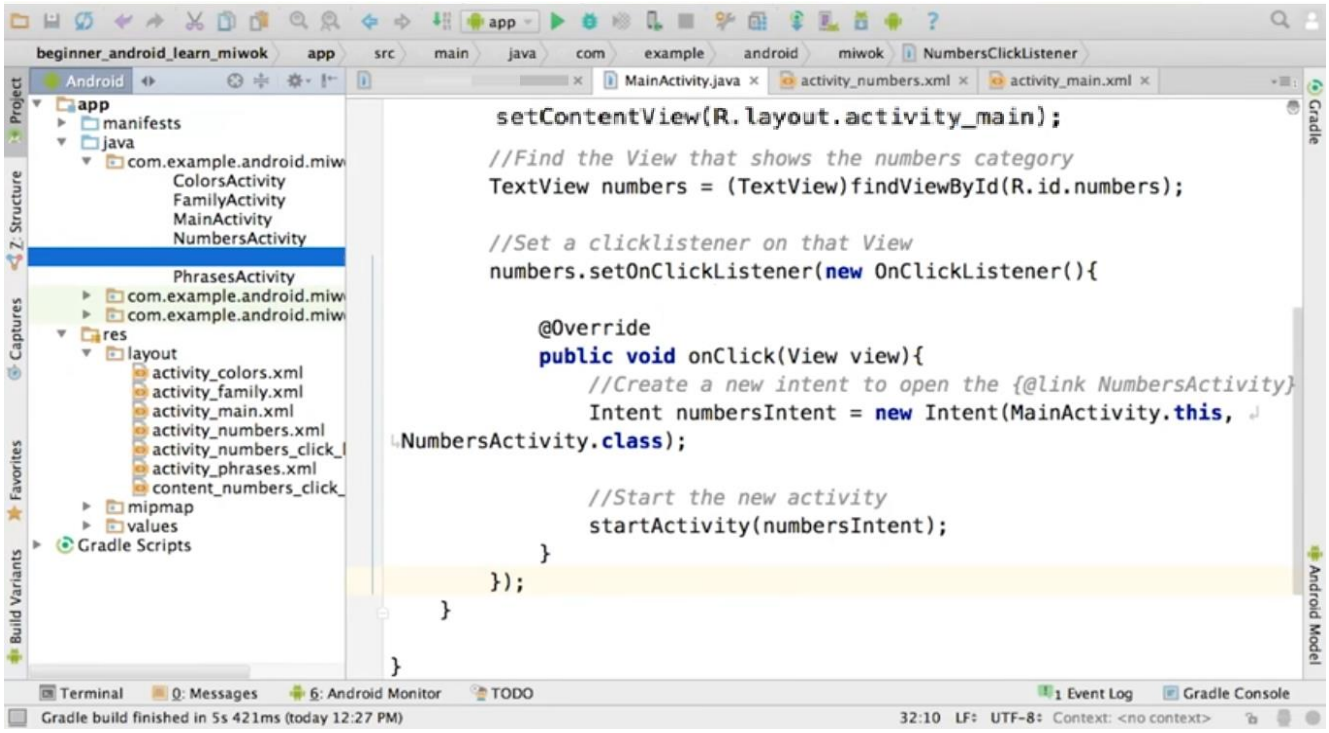
In MainActivity.java:

Attach the listener to the view

Create new object instance of event listener

Define the event listener inline (and the custom behavior for when the event happens).

```
buttonView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText(view.getContext(), "Open the list of numbers", Toast.LENGTH_SHORT).show();
    }
});
```



We are using Event Listeners because professional Android developer using them and they just need Java code.

USE ONCLICKLISTENERS FOR ALL CATEGORIES

1. Remove the old XML and Java code for handling the button click
2. Add the code **Provided below** to onCreate in the MainActivity file
3. Modify that code to repeat for all other activities
4. Modify the AndroidManifest.xml file so the app bar displays the name of the activity for – Numbers, Colors, Family Members, and Phrases

___ Check here when done 

12 app/src/main/AndroidManifest.xml

```
+        <activity
+            android:name=".FamilyActivity"
+            android:label="@string/category_family" />
+        <activity
+            android:name=".ColorsActivity"
+            android:label="@string/category_colors" />
+        <activity
+            android:name=".PhrasesActivity"
+            android:label="@string/category_phrases" />
     </application>

</manifest>
```

MainActivity.java

```
        // Find the View that shows the family category
    TextView family = (TextView) findViewById(R.id.family);



        // Set a click listener on that View
        family.setOnClickListener(new OnClickListener() {
            // The code in this method will be executed when the family category
is clicked on.
            @Override
            public void onClick(View view) {
                // Create a new intent to open the {@link FamilyActivity}
                Intent familyIntent = new Intent(MainActivity.this,
FamilyActivity.class);


                // Start the new activity
                startActivity(familyIntent);
            }
        });
```

Now that you have done event listeners for Numbers, Family, go and make event listeners for Colors and Phrases.

OnClickListener vs onClick

You might be wondering why we're going through all the trouble of creating an anonymous subclass of `OnClickListener` and attaching it to a view, when we already know how to use the `onClick` XML attribute from from back in Android Basics: User Input. Why write something terrifying like:

```
// In onCreate() in the Activity
Button button = (Button) findViewById(R.id.ze_button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        doSomeStuff();
    }
});
```

When we could do something much cleaner like:

```
android:onClick="myListener" // This is in the XML layout

public void myListener(View view){ // This is back in the Activity file
    doSomeStuff();
}
```

There are a couple reasons why you might want to programmatically set an `OnClickListener`. The first is if you ever want to change the behavior of your button while your app is running. You can point your button at another method entirely, or just disable the button by setting an `OnClickListener` that doesn't do anything.

When you define a listener using the `onClick` attribute, the view looks for a method with that name only in its host activity. Programmatically setting an `OnClickListener` allows you to control a button's behavior from somewhere other than its host activity. This will become very relevant when we learn about `Fragments`, which are basically mini activities, allowing you to build reusable collections of views with their own lifecycle, which can then be assembled into activities. Fragments always need to use `OnClickListeners` to control their buttons, since they're not Activities, and won't be searched for listeners defined in `onClick`.

Just to recap, now we became familiar with the existing code base, and then we tied Intents to each of our buttons, so we could move between activities in our App. Lastly we learned about on click listeners.

When to Use Arrays

Using Array will help us to store a number of variables and maintain the order of a list. The array is like a container with a fixed length, so once it is created it stays that length. An array can only store elements of the same type, so we have an entire array of strings, entire array of Boolean, entire array of integers. Try to answer this quiz.

Given these scenarios, what data structure would you use to store the data in the app? (integer, boolean, String, array of integers, array of Strings, etc.)

For a Travel app: Items to pack for traveling

For the YouTube app: Number of people who viewed a video

For the Settings app: Languages supported on an Android device

For a Shopping app: Whether a product is in stock or not

For a Lottery app: The winning combination of 5 numbers for a single round of the lottery

For a Health track app: Number of miles or kilometers walked for each day this week

Working with an Array

CREATE AND USE AN ARRAY

Create array

```
int [] shoeSizesAvailable = new int[3];
```

Initialize elements in an array

```
shoeSizesAvailable[0] = 5;
shoeSizesAvailable[1] = 7;
shoeSizesAvailable[2] = 10;
```

Access elements in an array

```
shoesSizesAvailable[0]    - -> value of 5
shoesSizesAvailable[1]    - -> value of 7
shoesSizesAvailable[2]    - -> value of 10
```

Get the array length

```
shoesSizesAvailable.length    - -> value of 3
```

WORKING WITH ARRAYS PRACTICE

soccerTeam

Which option of code provided in the gist would create the array above?

- Option 1
- Option 2
- Option 3
- Option 4

Option 1

```
int[] soccerTeam = new int[11];
soccerTeam[1] = 5;
soccerTeam[2] = 10;
soccerTeam[4] = 23;
soccerTeam[11] = 1;
```

Option 3

```
int[] soccerTeam = new int[11];
soccerTeam[0] = 5;
soccerTeam[10] = 1;
soccerTeam[4] = 23;
soccerTeam[1] = 10;
```

Option 2

```
int[] team = new int[11];
team[0];
team[1];
team[4];
team[10];
```

Option 4

```
Int [] soccerTeam = new Int [];
soccerTeam[0] = "5";
soccerTeam[1] = "10";
soccerTeam[4] = "23";
soccerTeam[10] = "1";
```

Now start to reflect what you have learned on your App by answering the following quiz and making the required changes to store numbers from one to ten.

ADD LIST OF WORDS TO THE APP (FOR EACH NUMBER 1 - 10)

What should the data type of the array be?

What should the size of the array be?

Add your code to the end of the onCreate() method in the NumbersActivity.java file:

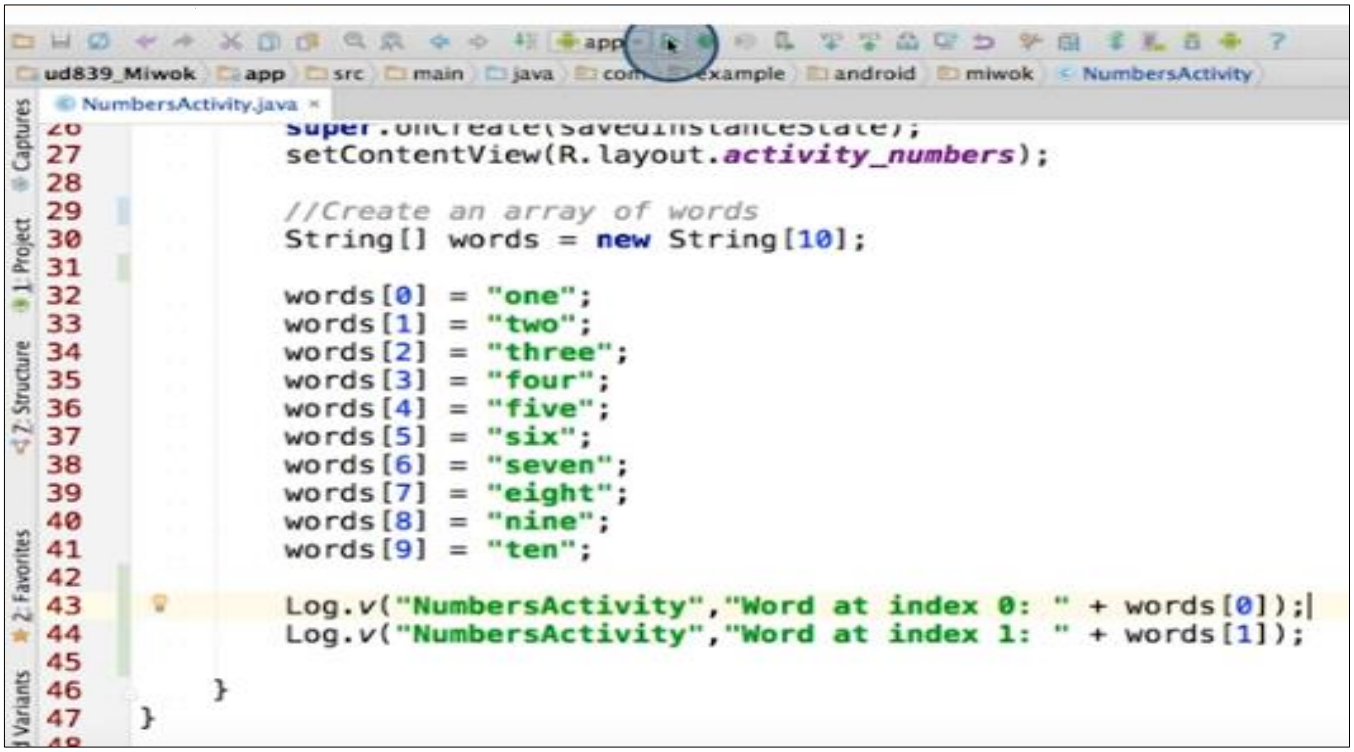
1. Create an array variable called "words"
2. Initialize the array with the numbers 1 - 10 ("one", "two", etc.)

13 app/src/main/java/com/example/android/miwok/NumbersActivity.java

```
@@ -24,5 +24,18 @@
24  24      protected void onCreate(Bundle savedInstanceState) {
25  25          super.onCreate(savedInstanceState);
26  26          setContentView(R.layout.activity_numbers);
    27  +
    28  +       // Create an array of words
    29  +       String[] words = new String[10];
    30  +       words[0] = "one";
    31  +       words[1] = "two";
    32  +       words[2] = "three";
    33  +       words[3] = "four";
    34  +       words[4] = "five";
    35  +       words[5] = "six";
    36  +       words[6] = "seven";
    37  +       words[7] = "eight";
    38  +       words[8] = "nine";
    39  +       words[9] = "ten";
27  40      }
28  41  }
```

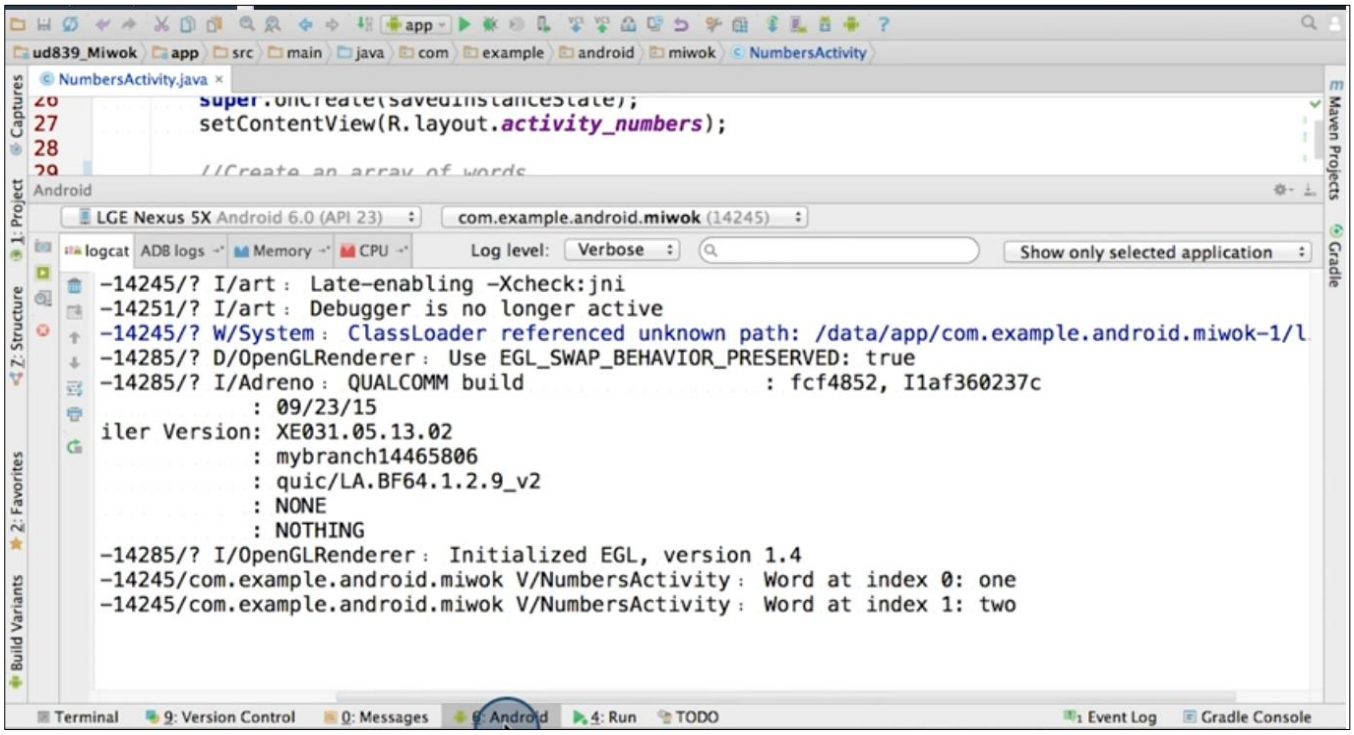
Verify Array Elements with Log Messages

We can use logging to check the state of variables while app is running, in this example two log messages are added to the bottom of onCreate method then run the App.



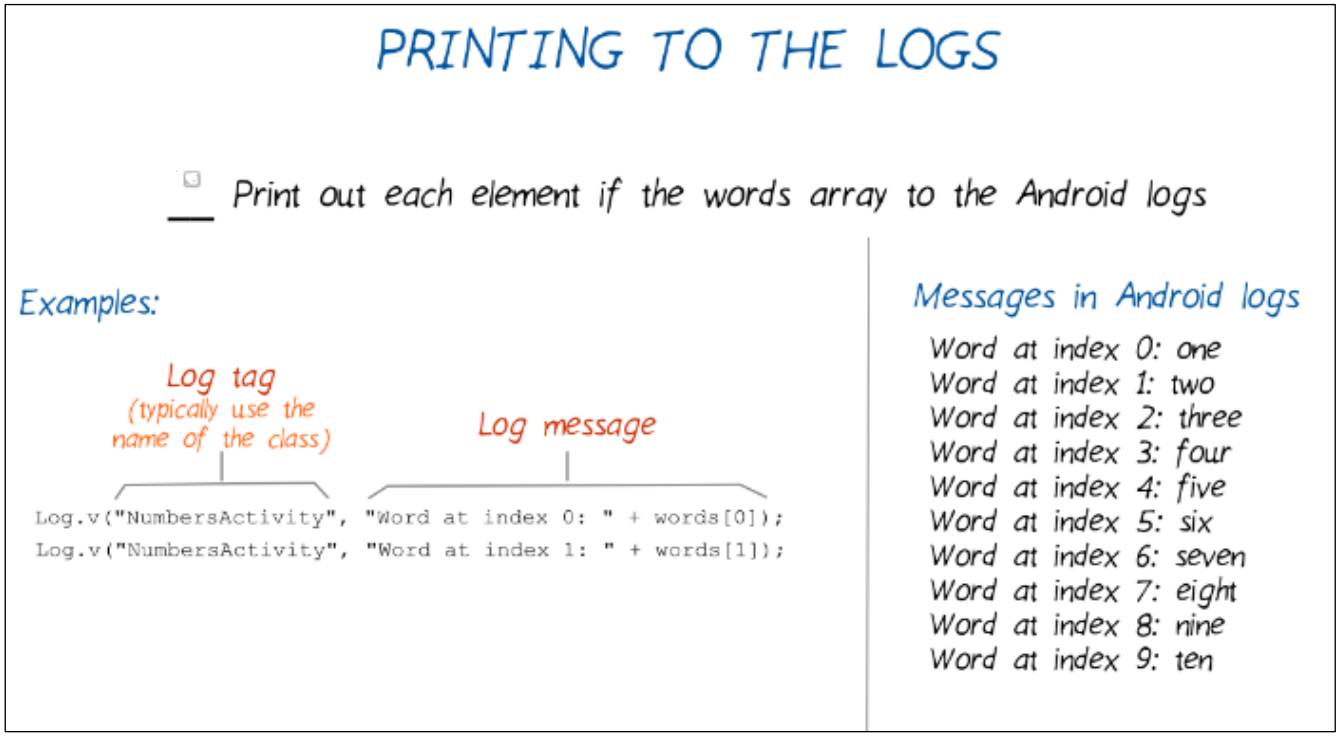
```
27        setContentView(R.layout.activity_numbers);
28
29        //Create an array of words
30        String[] words = new String[10];
31
32        words[0] = "one";
33        words[1] = "two";
34        words[2] = "three";
35        words[3] = "four";
36        words[4] = "five";
37        words[5] = "six";
38        words[6] = "seven";
39        words[7] = "eight";
40        words[8] = "nine";
41        words[9] = "ten";
42
43        Log.v("NumbersActivity","Word at index 0: " + words[0]);
44        Log.v("NumbersActivity","Word at index 1: " + words[1]);
45
46    }
47}
```

Now with the App running, we navigate to the number activity and notice that these two messages in the log output appear.



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

-14245/? I/art: Late-enabling -Xcheck:jni
-14251/? I/art: Debugger is no longer active
-14245/? W/System: ClassLoader referenced unknown path: /data/app/com.example.android.miwok-1/l
-14285/? D/OpenGLRenderer: Use EGL_SWAP_BEHAVIOR_PRESERVED: true
-14285/? I/Adreno: QUALCOMM build                   : fcf4852, I1af360237c
            : 09/23/15
iler Version: XE031.05.13.02
            : mybranch14465806
            : quic/LA.BF64.1.2.9_v2
            : NONE
            : NOTHING
-14285/? I/OpenGLRenderer: Initialized EGL, version 1.4
-14245/com.example.android.miwok V/NumbersActivity: Word at index 0: one
-14245/com.example.android.miwok V/NumbersActivity: Word at index 1: two
```

At this moment, it your time to practice with logging messages by trying this quiz:



Working with An ArrayList

The problem is that a limitation with arrays, in case we want to add more elements in the future. This data structure is called ArrayList. An array list can hold object data types not primitive data types. But if we want to store primitive data types in array list , in Java we have something called **object grabbers** they wrap around primitive data types to help them become object data type.

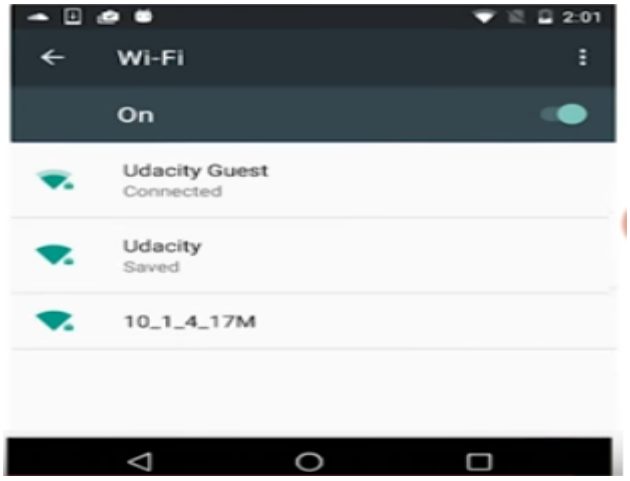
ArrayList<Integer>✓

ArrayList<int>✗

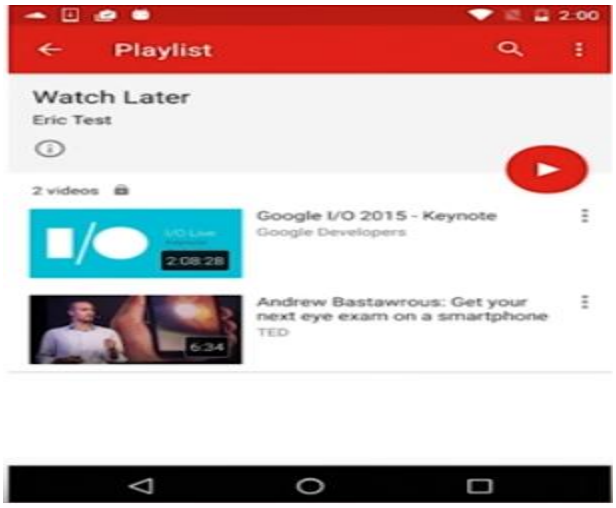
	Array	ArrayList
Can change size once created?	No	Yes
Is a class?	No	Yes
Uses methods to access and modify elements?	No	Yes
What can it store?	Primitive and Objects	Only Objects

Let's take some scenarios on using array and array list in real Apps.

For Wi-Fi screen, we probably store information about available Wi-Fi networks in an array list, and this because the list of available networks can grow or shrink based on where you're located. Some time there are no networks besides you, and other times there might be too many, in this case, it doesn't make sense to have a fixed size array.

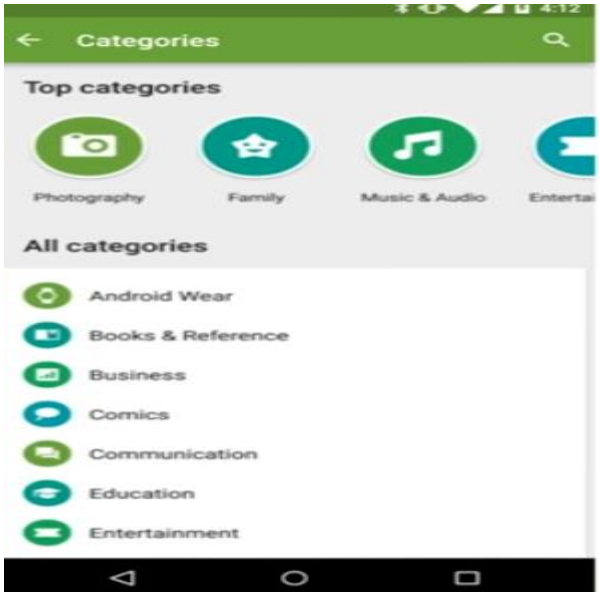


In the YouTube app, there's a feature that allows you to create a playlist of videos to watch later, and this is an ideal situation for array list because we will continue to add videos that we want to see and remove that those already watched, and since we need the data structure to dynamically change the size, we should use an array list



In the alarm clock App, we have the potential to use an array. When you specify an alarm you specify what day of the week the alarm should repeat on. This could be a Boolean array, which source true if the alarm would happen on that day and false if the alarm should not happen that day. So there are only seven days in a week, so the size of our array won't change with the time. Hence we can use an array.

For the categories of Apps in play store we can also use an array, since the number of categories is fixed over time.



In the comparison table you saw, you can notice that an ArrayList is a **class** and you need to call methods to access and modify elements of the list.

HOW TO CREATE AND ACCESS ELEMENTS IN AN ARRAYLIST

Create an ArrayList

```
ArrayList<String> musicLibrary = new ArrayList<String>();
```

Add elements in an ArrayList

```
musicLibrary.add("Yellow Submarine");
musicLibrary.add("Thriller");
// Adds an element at a specific index
musicLibrary.add(0, "Blue Suede Shoes");
```

Access elements in an ArrayList

```
musicLibrary.get(0);
musicLibrary.get(1);
musicLibrary.get(2);
```

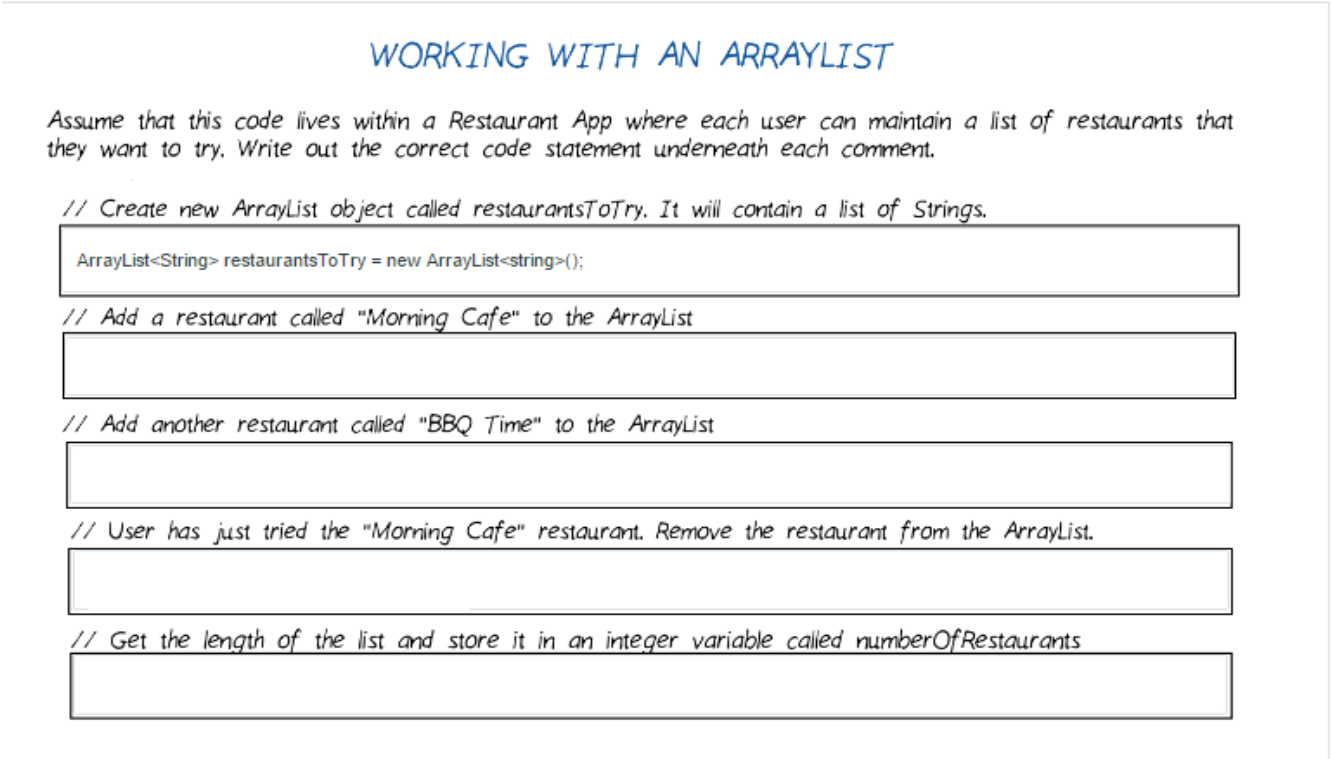
Remove elements from an ArrayList

```
// Removes the element at the specific index
musicLibrary.remove(2);
```

Get the ArrayList length or size

```
musicLibrary.size();
```

Now you can try this quiz to improve your understanding of ArrayList.



Type Parameter Naming Conventions

By convention, type parameter names are single, uppercase letters. This stands in sharp contrast to the variable naming conventions that you already know about, and with good reason: Without this convention, it would be difficult to tell the difference between a type variable and an ordinary class or interface name.

The most commonly used type parameter names are:

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value
- S,U,V etc. - 2nd, 3rd, 4th types

You'll see these names used throughout the Java SE API and the rest of this lesson.

Switch from Array to ArrayList

	Array	ArrayList
Create	`String[] names = new String[2];`	`ArrayList<String> names = new ArrayList<String>();`

Modify Element	`names[0] = "Jessica";`
	`names[1] = "Chris";`
		`names.add("Jessica");`
		`names.add("Chris");`
		`names.remove("Chris");`

Access Element	`names[0]` `names[1]`	`names.get(0);` `names.get(1);`
Size	`names.length`	`names.size();`

USE ARRAYLIST IN NUMBERSACTIVITY

Modify the NumbersActivity file:

- ☐ Instead of storing the words in an array, store them using an ArrayList of Strings. The ArrayList should have the same variable name called "words".
- ☐ Update the logging statements to print out each element of the list

The words should be the same as before:

one
two
three
four
five
six
seven
eight
nine
ten

```
// create a list of words

+        ArrayList<String> words = new ArrayList<String>();
+        words.add("one");
+        words.add("two");
+        words.add("three");
+        words.add("four");
+        words.add("five");
        words.add("six");   words.add("seven");    words.add("eight");
words.add("nine");   words.add("ten");
```

```
// verify the contents of the list by printing out each element to the logs
+        Log.v("NumbersActivity", "Word at index 0: " + words.get(0));
+        Log.v("NumbersActivity", "Word at index 1: " + words.get(1));
+        Log.v("NumbersActivity", "Word at index 2: " + words.get(2));
+        Log.v("NumbersActivity", "Word at index 3: " + words.get(3));
+        Log.v("NumbersActivity", "Word at index 4: " + words.get(4));
Log.v("NumbersActivity", "Word at index 5: " + words.get(5));
Log.v("NumbersActivity", "Word at index 6: " + words.get(6));
Log.v("NumbersActivity", "Word at index 7: " + words.get(7));
Log.v("NumbersActivity", "Word at index 8: " + words.get(8));
Log.v("NumbersActivity", "Word at index 9: " + words.get(9));
```

Add Text Views to the Layout Based on the List

To add TextView for the list, we should give and ID for the **activity_numbers.xml**, then we start changing it in Java, as seen in the code below:

NumbersActivity.java × activity_numbers.xml ×

```
29          words.add("ten");
30
31          LinearLayout rootView = (LinearLayout)findViewById(R.id.rootView);
32
33          int index = 0;
34
35          TextView wordView = new TextView(this);
36          wordView.setText(words.get(index));
37          rootView.addView(wordView);
38
39          index = index + 1;
40
41          TextView wordView2 = new TextView(this);
42          wordView2.setText(words.get(index));
43          rootView.addView(wordView2);
44
45          index = index + 1;
46
47          TextView wordView3 = new TextView(this);
48          wordView3.setText(words.get(index));
49          rootView.addView(wordView3);
50
51
52
53      }
```

This way of writing many blocks of codes for displaying elements of ArrayList is inefficient, and now we are going to use a programming concept called **Loops**. Then will change our code to use loops.

Intro to Loops

WHILE LOOP

```
int index = 0;

while(index < 4) {

  Log.v("NumbersActivity",
  "Index: " + index + "
  Value: " + words[index]);

  index++;
}
```

1. What is the task to do each time?
   Print out a log statement for a specific item in the list (based on the index variable)
2. How many times to repeat the loop?
   4 times
3. What is the condition?
   Index variable < 4 (assuming index starts at 0 and increments by 1 each time)

USE THE WHILE LOOP

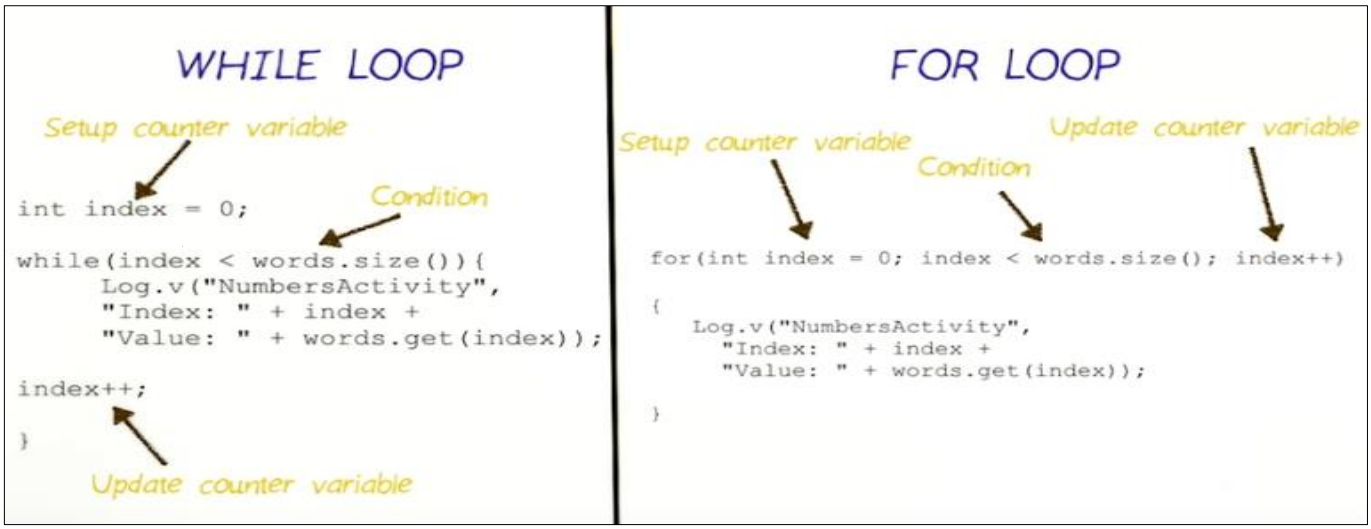
- 1. Remove log messages in NumbersActivity onCreate() method.
- 2. Modify activity_numbers.xml to:
  - Change the RelativeLayout into a LinearLayout
  - Set LinearLayout orientation to be vertical
  - Add a view ID called rootView
- 3. Modify NumbersActivity onCreate() method so that the code uses a `while` loop to create and display a TextView for each element in the words list.

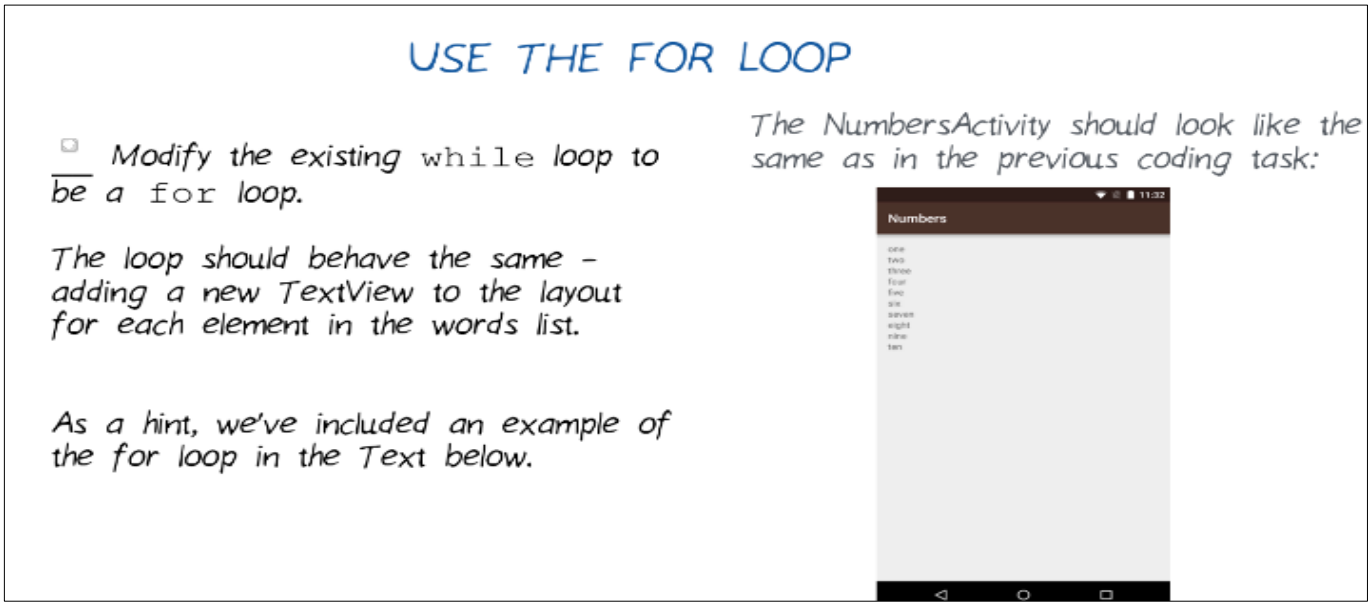
  You can start where we last left off with the code for the NumbersActivity, which creates 3 TextViews manually (without a loop).

After steps 1 – 3, NumbersActivity in your app should look like this:

Here the code for while loop:

```
  // Find the root view so we can add child views to it

        LinearLayout rootView = (LinearLayout) findViewById(R.id.rootView);
        // Create a variable to keep track of the current index position
        int index = 0;
        // Keep looping until we've reached the end of the list (which means keep
looping
        // as long as the current index position is less than the length of the
list)        while (index < words.size()) {            // Create a new TextView
            TextView wordView = new TextView(this);
            // Set the text to be word at the current index
            wordView.setText(words.get(index));
            // Add this TextView as another child to the root view of this layout
            rootView.addView(wordView);
            // Increment the index variable by 1
            index++;        }    }   }
```





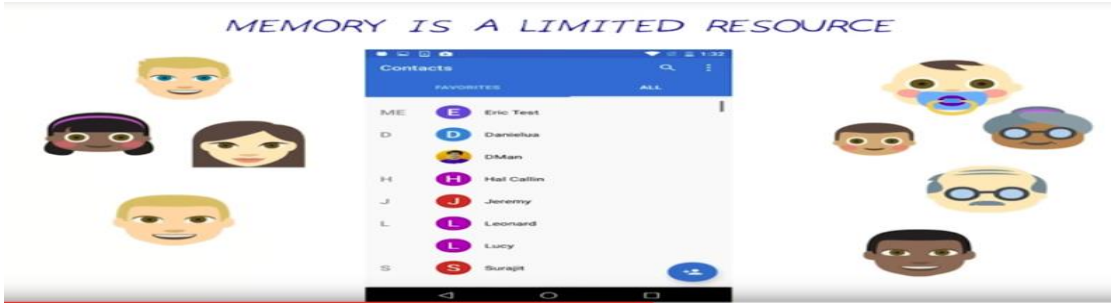

```
// Find the root view so we can add child views to it

        LinearLayout rootView = (LinearLayout) findViewById(R.id.rootView);
  // Keep looping until we've reached the end of the list (which means keep looping
    // as long as the current index position is less than the length of the list).
        // The index variable keeps track of our current position in the list and
        // increments by 1 each time the code in the loop is executed.
        for (int index = 0; index < words.size(); index++) {
            // Create a new TextView
            TextView wordView = new TextView(this);
            // Set the text to be word at the current index
            wordView.setText(words.get(index));
            // Add this TextView as another child to the root view of this layout
            rootView.addView(wordView);      }
    }
}
```

Memory is a Limited Resource

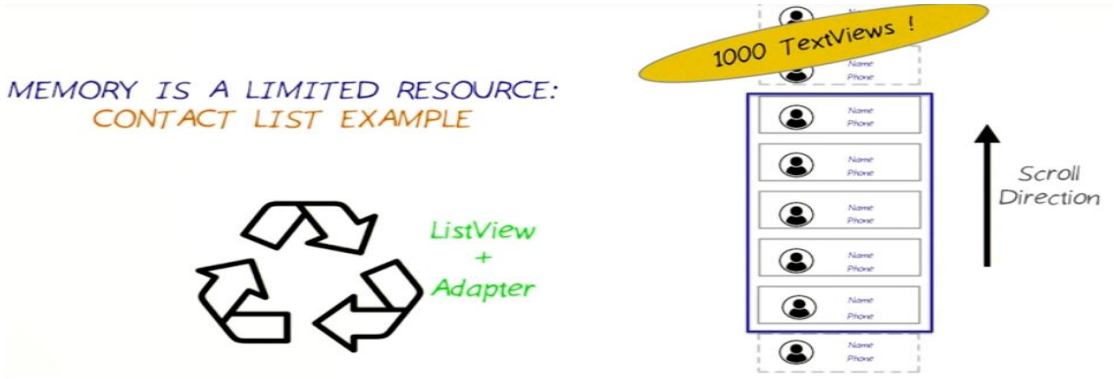
As developers we should keep on our minds that memory is a limited resource.

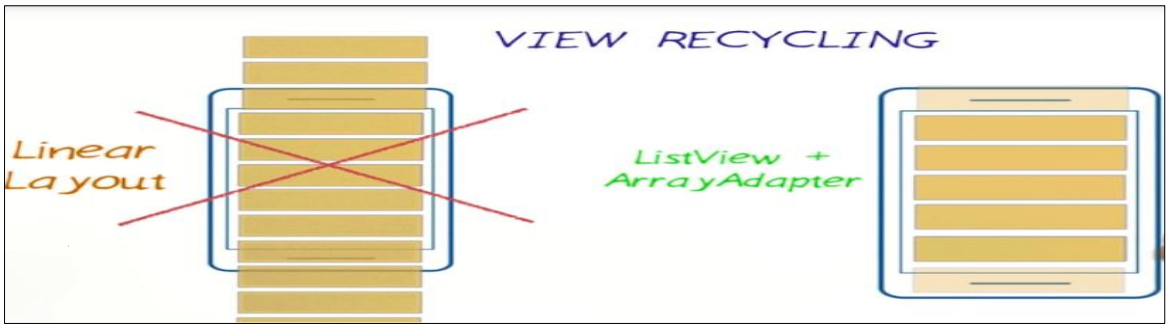
So if one of these apps or our app consumes a lot of memory that can have a significant impact on running all other apps. And the bad thing it will negatively affect the overall experience of the entire device, which could lead to a very frustrating user experience. So we need to create apps that use this memory efficiently.

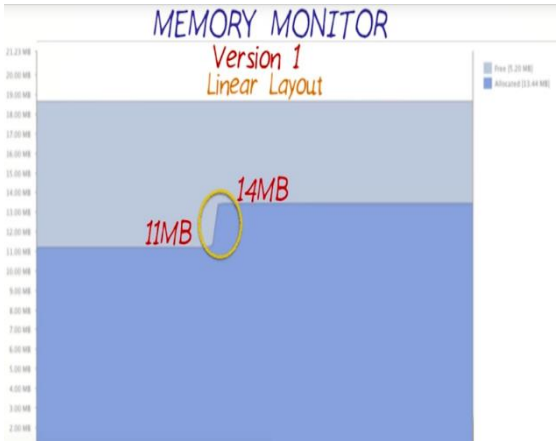
Consider the contacts list app, a user may have 1000 names and phone numbers stored into their contacts list. Now, if each contact were stored as TextViews in linear layout, the phone would expense a lot of memory creating, displaying 1000 TextViews. Even though may be not all 1000 would be shown on the screen at the same time.

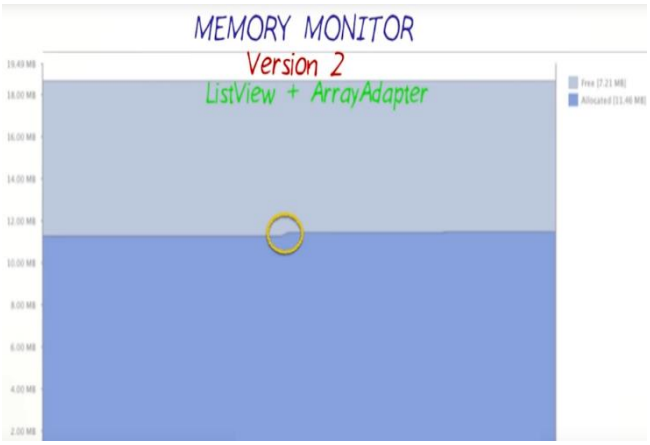


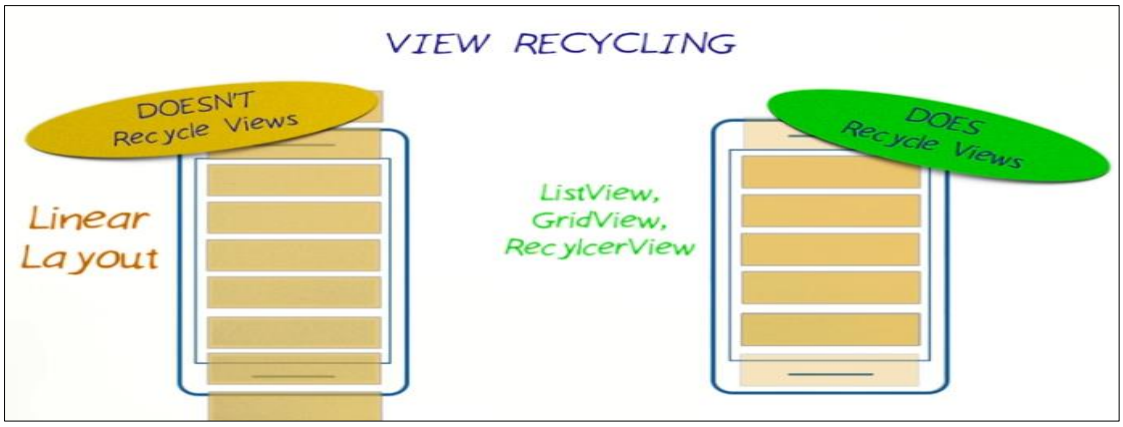
If a large part of the system's memory goes into creating and storing 1000 contact, then there is not much left to run other parts on the phone. And this may cause to other apps to not run at all. To solve this issue we could reuse 5 views to show the 1000 TextView.







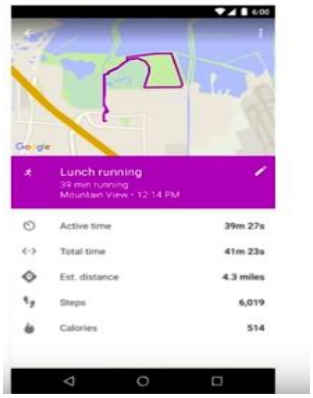




There are some cases that need to recycle views , and others where no need for recycling views.

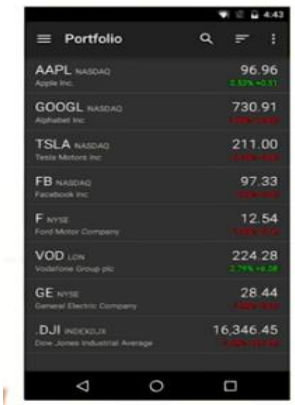
In TWITTER app there are millions of tweets , so we can recycle views .

GOOGLE FIT APP



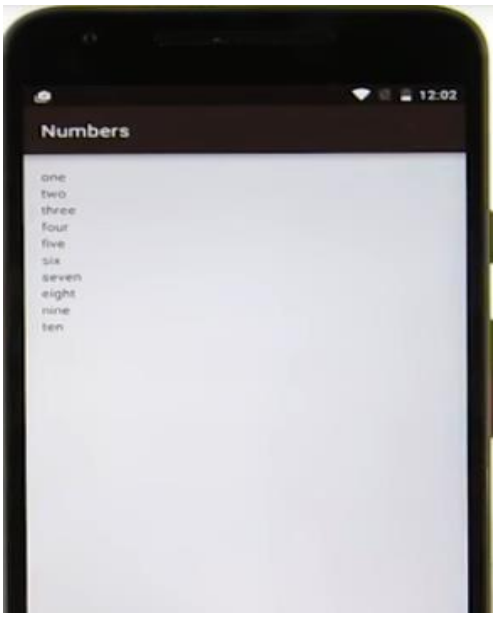
In the Google fit app, there is a page listing few fixed number of details, so there is no need to recycle views. This portion of the app can use vertical linear layout.

STOCKS APP



For those who are interested in finance, it is quite common for users to have a lot of companies that they're interested in. So the list could get really long, so the app can display all these information about these companies by recycling views as the user is scrolling through the list.

Now we go back to our app, there's a few fixed categories here, and this list is pretty fixed, so in this case using a vertical linear layout with four text views. But when we open the Numbers Activity, these list items can get more complicated by display the numbers 1 through 1000. So this list is a good use case for view recycling.



Let's see how to modify our code so that we can use the list view and array adapter.

Modify the NumbersActivity onCreate() method:

1. Remove the for loop which created a new TextView and added it to the layout for each word in the list.
2. Add these 3 lines of code to the method, after the words list has been created:

```
    ArrayAdapter<String> itemsAdapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, words);

    ListView listView = (ListView) findViewById(R.id.list);

    listView.setAdapter(itemsAdapter);
```

To see what your NumbersActivity.java file should look like after this change, check out the github link here.

Modify the activity_numbers.xml layout file: -

3 . Replace the whole file with this XML. It contains a single ListView element with view ID list.

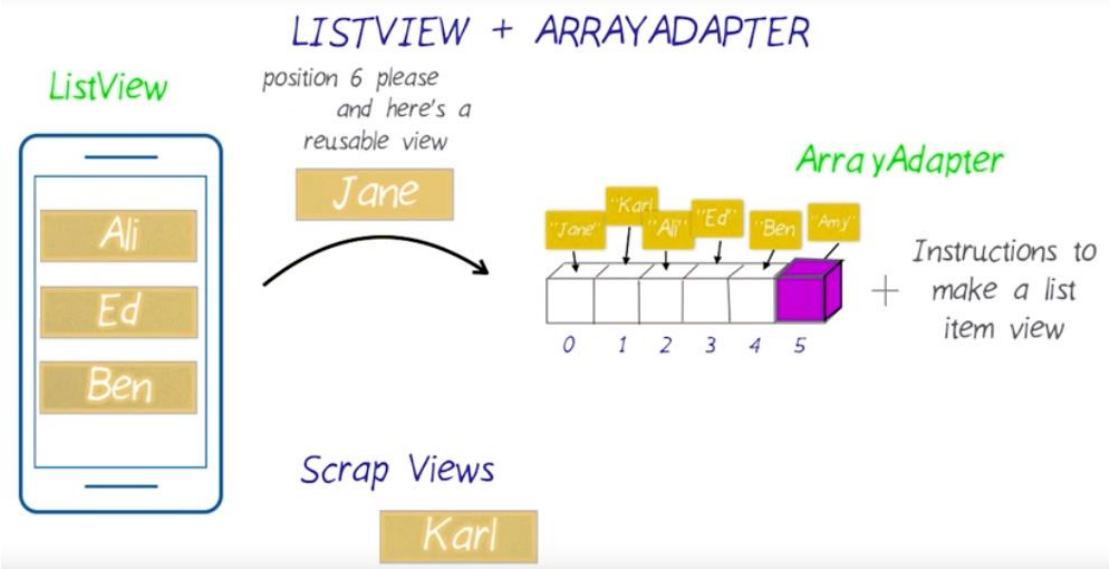
```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.android.miwok.NumbersActivity"/>
```

Github code for NumbersActivity.java file:

```
public class NumbersActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_numbers);

        // Create a list of words
        ArrayList<String> words = new ArrayList<String>();
        words.add("one");
        words.add("two");
        words.add("three");
        words.add("four");
        words.add("five");
        words.add("six");
        words.add("seven");
        words.add("eight");
        words.add("nine");
```

```
        words.add("ten");

        // Create an {@link ArrayAdapter}, whose data source is a
list of Strings. The
        // adapter knows how to create layouts for each item in the
list, using the
        // simple_list_item_1.xml layout resource defined in the
Android framework.
        // This list item layout contains a single {@link TextView},
which the adapter will set to
        // display a single word.
        ArrayAdapter<String> itemsAdapter =
new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
words);

        // Find the {@link ListView} object in the view hierarchy of
the {@link Activity}.
        // There should be a {@link ListView} with the view ID called
list, which is declared in the
        // activity_numbers.xml layout file.
        ListView listView = (ListView) findViewById(R.id.list);

        // Make the {@link ListView} use the {@link ArrayAdapter} we
created above, so that the
        // {@link ListView} will display list items for each word in
the list of words.
        // Do this by calling the setAdapter method on the {@link
ListView} object and pass in
        // 1 argument, which is the {@link ArrayAdapter} with the
variable name itemsAdapter.
        listView.setAdapter(itemsAdapter);
    }
}
```

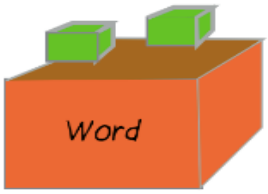
Without the adapter the ListView is just an empty container. This ListView is powered by ArrayAdapter that holds onto the set of data shown on the screen. So the ArrayAdapter adapt data into a list item view that will be displayed in the ListView.



Create the Word Class

DATA STRUCTURE QUESTION

Plan how to create the Word class for the Miwok app, so we can show a list of English and Miwok words on the screen for the numbers 1-10.



What state should a Word object contain? What is the data type of each variable?

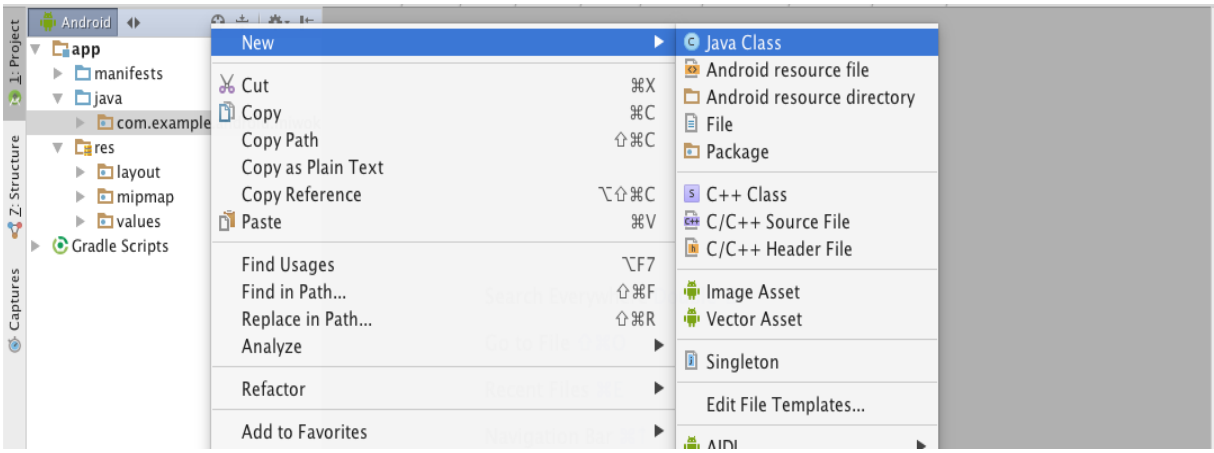
```
2 String variables for storing Miwok and English  words
```

What methods should we have?

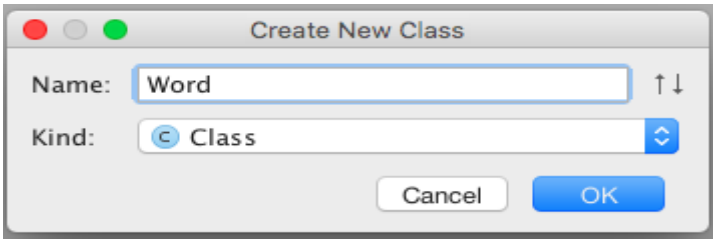
```
A method to get English translation, and a method to get Miwok translation
```

1. Create a new file called "Word.java" and add it under the "com.example.android.miwok" folder.

To create a new Class in Android Studio. Right click on the package in the project directory. Go to new Class.



When the wizard pops up, give the class a name: Word. Accept the defaults and click Finish.



2. Define the Word class in the Word.java file with state and methods.

There should be no change in the appearance of your app yet, but be sure to run the app on your device to verify there are no errors in your code.

Now finish implementing the WordAdapter class by looking at the sample Android Flavors app **hosted on the server .**

Remaining Words

Adding the Remaining Words

You've successfully added a ListView and a custom ArrayAdapter to your app to display the English and Miwok words for the Numbers category.

Step 1:

Finish modifying the ColorsActivity, FamilyActivity, and PhrasesActivity so we see the English and Miwok translations for all words in the app.
We don't have images yet, but that will come in the next step.

Hint #1: To get started with the ColorsActivity, instead of writing all the list code again from scratch, start off by copying the NumbersActivity code over to the ColorsActivity file. We'll use this as a base and then modify as needed. Rename the class to be called ColorsActivity (so it matches the name of the file).

Step: 2

Run it on the device to make sure the ColorsActivity is displaying something - even though it's still the wrong vocab words. We can switch them later.

Step: 3

Ok, we know the code works, we just need to swap out the numbers with the color words. You can repeat the copy/pasting for the other activities as well. Run your app again to make sure everything worked.

Hint #2: To save yourself some extra work, we can reuse the activity_numbers.xml for all 4 activities, instead of updating activity_colors.xml, activity_family.xml, and activity_phrases.xml to also have a ListView XML element. The activity_numbers.xml has nothing specific to numbers in it because all the words are added to the list dynamically when the app is running via Java code.

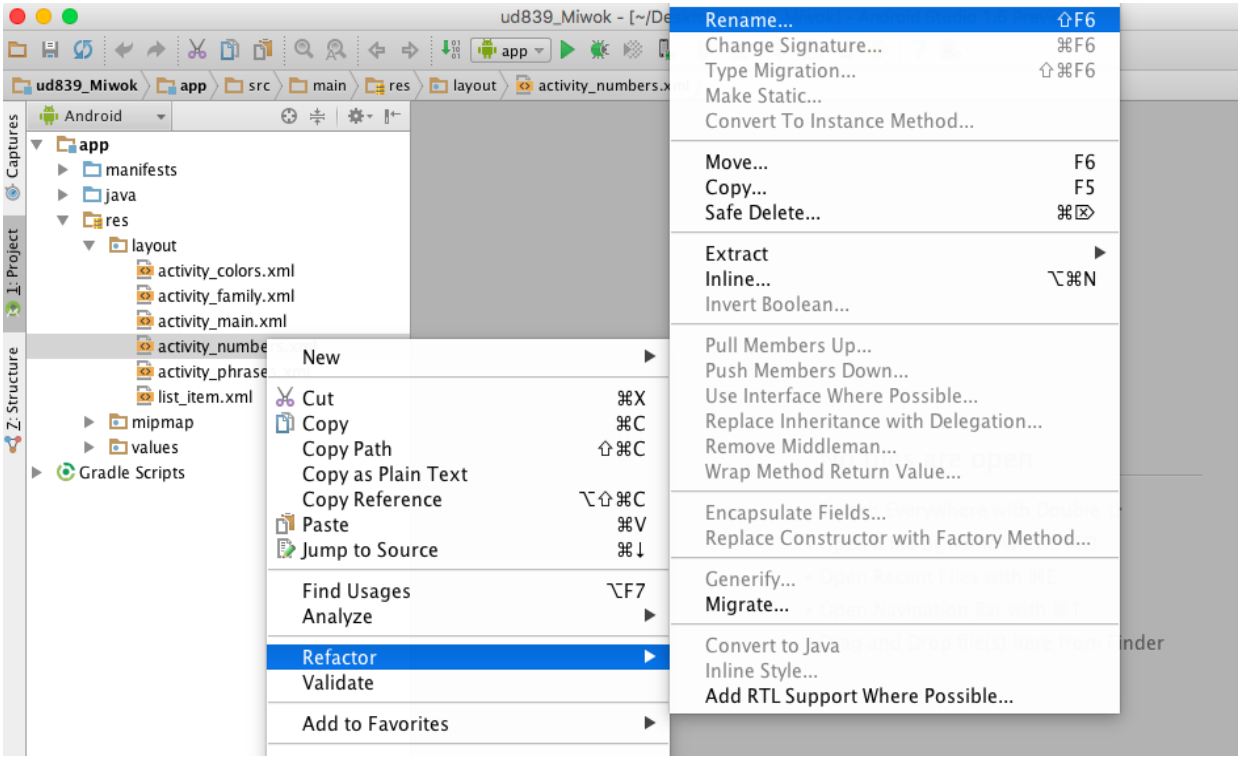
activity_numbers.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:tools="http://schemas.android.com/tools"
 android:id="@+id/list"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 tools:context="com.example.android.miwok.NumbersActivity"/>
```

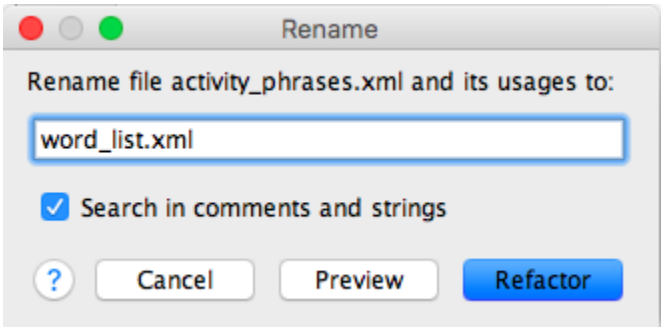
Remember the tools attribute is Numbers-specific, so we can remove that.

Let's just pick a better name for the activity_numbers.xml file to make it more general to all the categories (not just the Numbers category). How about renaming the file to word_list.xml?

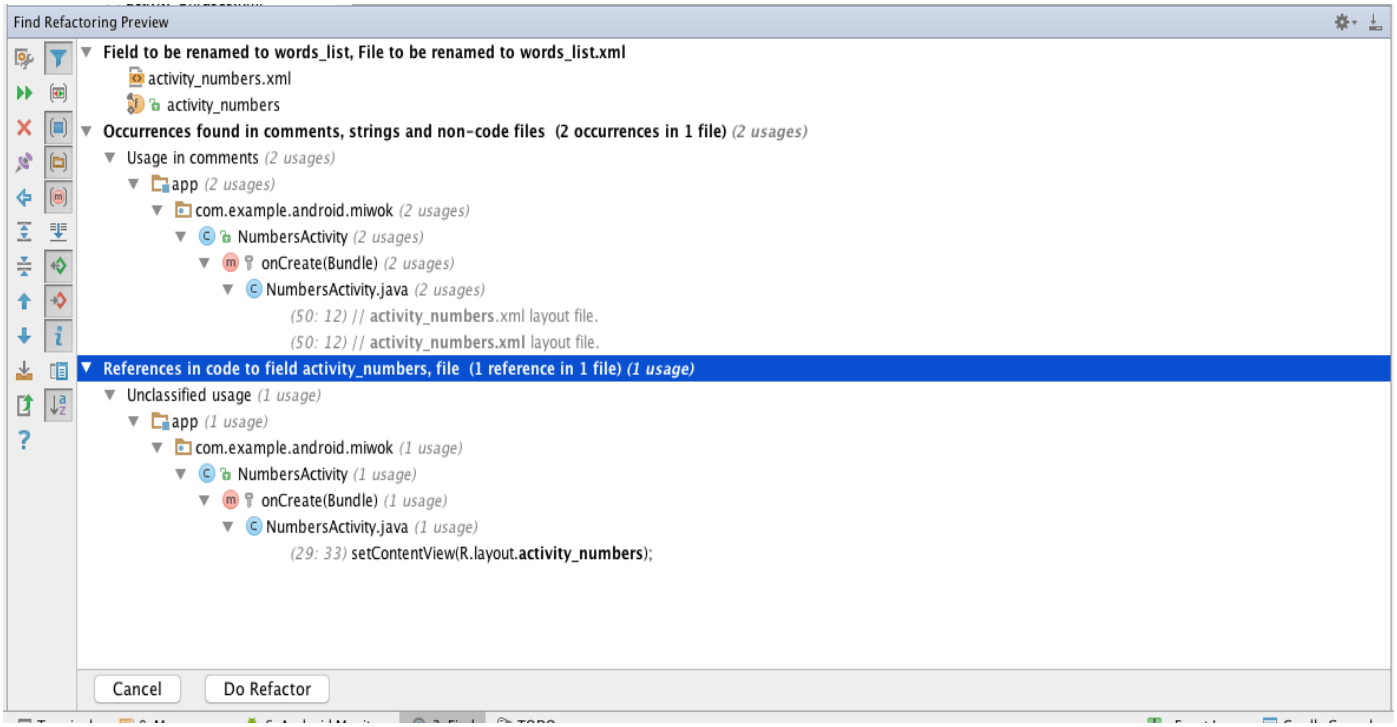
Select activity_numbers.xml -> Refactor -> Rename:



Rename to word_list.xml:



Then click the "Do Refactor" button. All occurrences of activity_numbers will be replaced with the new name: word_list:



Step: 4

Now we can set the content view of each activity to be word_list layout.

Example:

```
public class ColorsActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.word_list);
```

Be sure to delete any unused layout files such as activity_colors.xml, activity_phrases.xml, and activity_family.xml.