Copyright:
Author: Daniel DANG
School of Computing (SoC), Eastern Institute of Technology (EIT)
Email: daniel.dang.nz@gmail.com

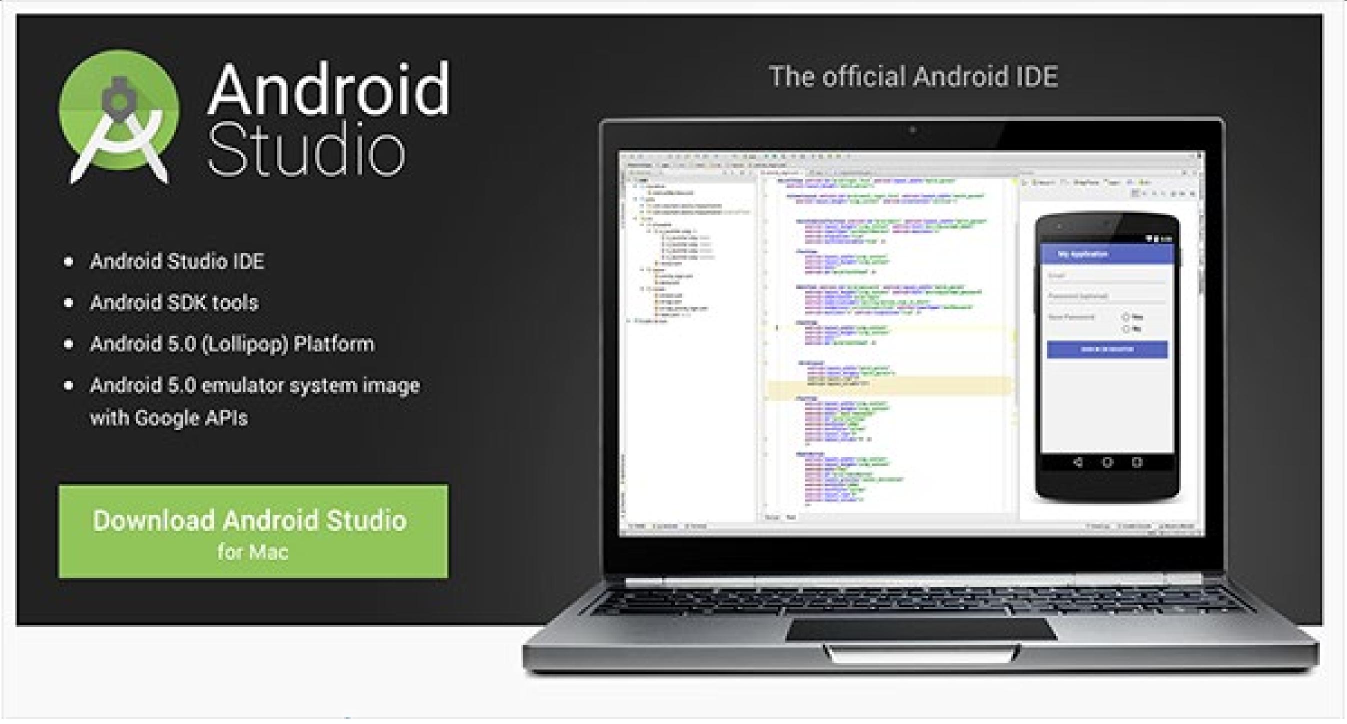
PRACTICAL LAB 1:
Create Graphic User Interface (GUI) in Android

1. Objectives:

The preparatory work for this lab has already been completed by us to save time, including **Android studio IDE** & JDK installation. Students will learn how to create, build, run, and debug the first application in Android Studio.

In this lab, students will develop **a simple layout (screen)** including an **ImageView (display college logo)**, three **TextViews (display strings)**, and **a TextClock showing current time**. Furthermore, upon completion, students also learn:

Design Graphic User Interface (GUI): - **Layout**: Relativelayout - **Background**: use color, xml background, or image - **Visual Controls**: TextView (display strings/text), ImageView (display image), EditText, Button - **Set TextView/ImageView properties**: text, background, style, size, fontFamily - **App style & theme, and "app_name"** - **XML View properties**: padding, margin, gravity, match_parent, wrap_content, ...	**Java coding:** - **Use Toast class** to pop up a message on screen; - **Set Click Listener** for Visual controls; - **Use AlertDialog class** to add an **Alert Dialog** to ask user confirm "Exit" operation; **Best practices:** - Create ***color variables*** in **colors.xml file** - Create ***string variables*** in **strings.xml file** - Create ***background drawable*** in **/drawable/ folder** - Create ***menu's items*** in **/menu/ folder**


Android Studio
The official Android IDE
Android Studio IDE
Android SDK tools
Android 5.0 (Lollipop) Platform
Android 5.0 emulator system image with Google APIs
Download Android Studio for Mac

Android Studio IDE

Mobile App Development Process

Step 1: Create a new Android Project
+ Target devices
+ Create an Activity & associated layout (screen)

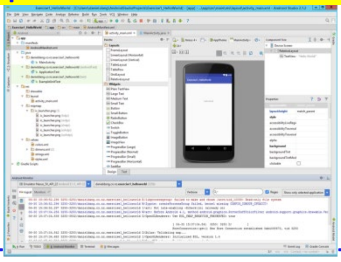
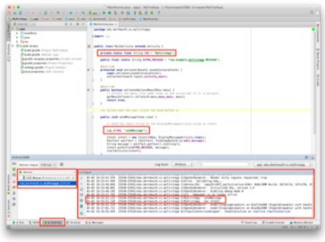
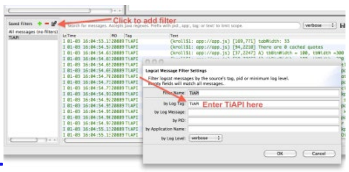
Step 2: Create Android Resources
+ res/drawables: images, shapes, animations,
+ res/values/colors.xml: define a list of colors for your app
+ res/values/strings.xml, res/values/dimens.xml, etc.

Step 3: Design graphic UI (res/layout) – XML layout
+ Design layout frame (nested layout)
+ Add Widget controls (TextView, Button, EditView, etc.)

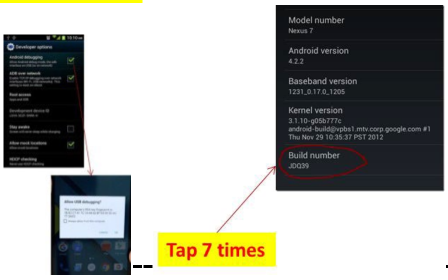
Step 4: Java coding – Activities and other java classes
+ Import classes, interfaces, and pacakges; Declare global objects, variables (right after class declaration)
+ Edit **onCreate() method** to create your Activity & its associated screen), to do casting/referencing (findViewID)
+ Edit **onClick() method** to respond to user events (clicks);

Step 5: Update AndroidManifest
+ App version, SDK target
+ Orientation
+ Permissions

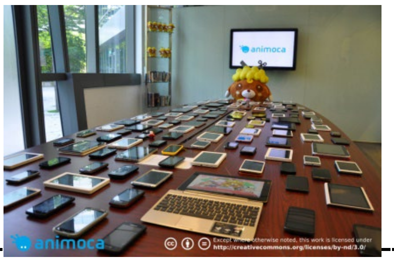
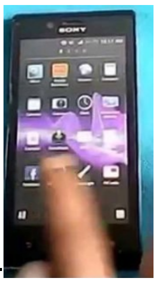
Step 6: Debug the app with Android Virtual Devices (AVD) & Logcat (DDMS)

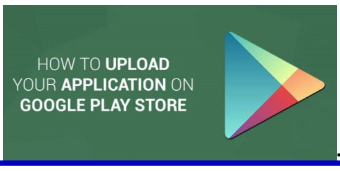
Step 7: Test on real devices

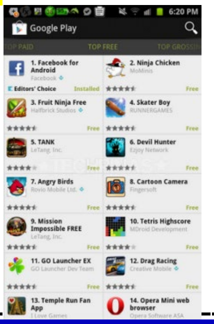


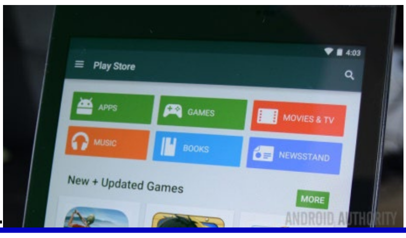
Model number
Nexus 7
Android version
4.2.2
Baseband version
1231_0.17.0_1205
Kernel version
3.1.10-g05b777c
android-build@vpbs1.mtv.corp.google.com #1
Thu Nov 29 10:35:37 PST 2012
Build number
JDQ39


animoca
Except where otherwise noted, this work is licensed under http://creativecommons.org/licenses/by-nd/3.0/


Step 8: Upload the app onto Google App Store







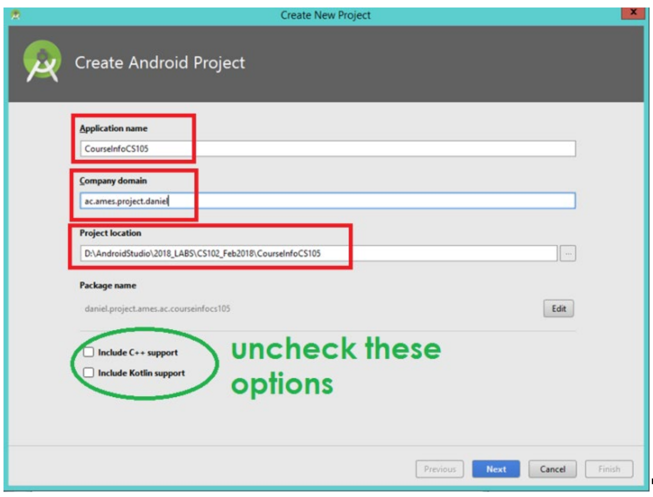
PART 1: CREATE A NEW ANDROID PROJECT

Open Android Studio IDE and create a new project as below:

Step 1: Create a new Android project

- Application name: **WelcomeToCS102_[yourname]**
- Company Domain: **ac.ames.project. [yourname]**
- Target Android devices: **Phone and Tablet**;
  - Minimum SDK: **API 21: Android 5.0 (Lollipop)**
- Add an Activity to Mobile: **Empty Activity**;
  - Activity name: **MainActivity**
  - Layout name: **activity_main**

+ Target devices: run on 71.3% devices
_Smartphone & Tablet
_Android KitKat (5.0) & API21 (min)



Application name: CourseInfoCS105

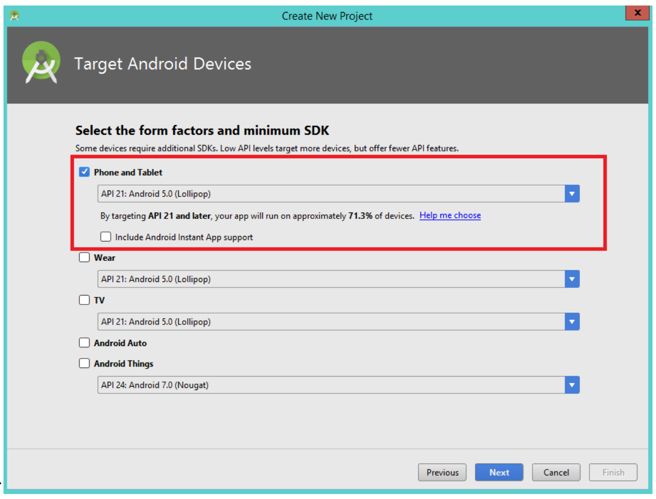
Company domain: ac.ames.project.daniel

Project location: D:\AndroidStudio\2018_LABS\CS102_Feb2018\CourseInfoCS105

Package name: daniel.project.ames.ac.courseinfocs105

☐ Include C++ support
☐ Include Kotlin support

uncheck these options



Select the form factors and minimum SDK

Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

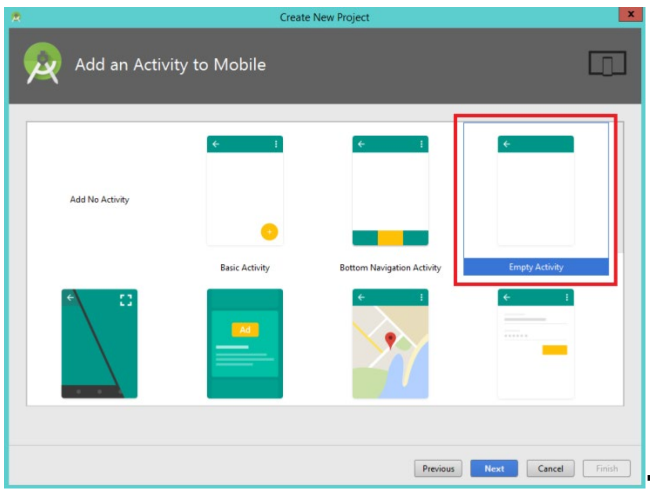
☑ Phone and Tablet
API 21: Android 5.0 (Lollipop)
By targeting **API 21 and later**, your app will run on approximately **71.3%** of devices.
☐ Include Android Instant App support

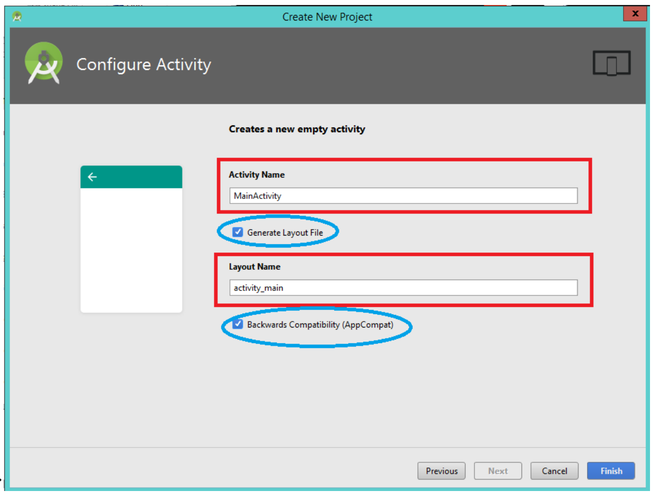
☐ Wear
API 21: Android 5.0 (Lollipop)

☐ TV
API 21: Android 5.0 (Lollipop)

☐ Android Auto

☐ Android Things
API 24: Android 7.0 (Nougat)





Creates a new empty activity

Activity Name: MainActivity

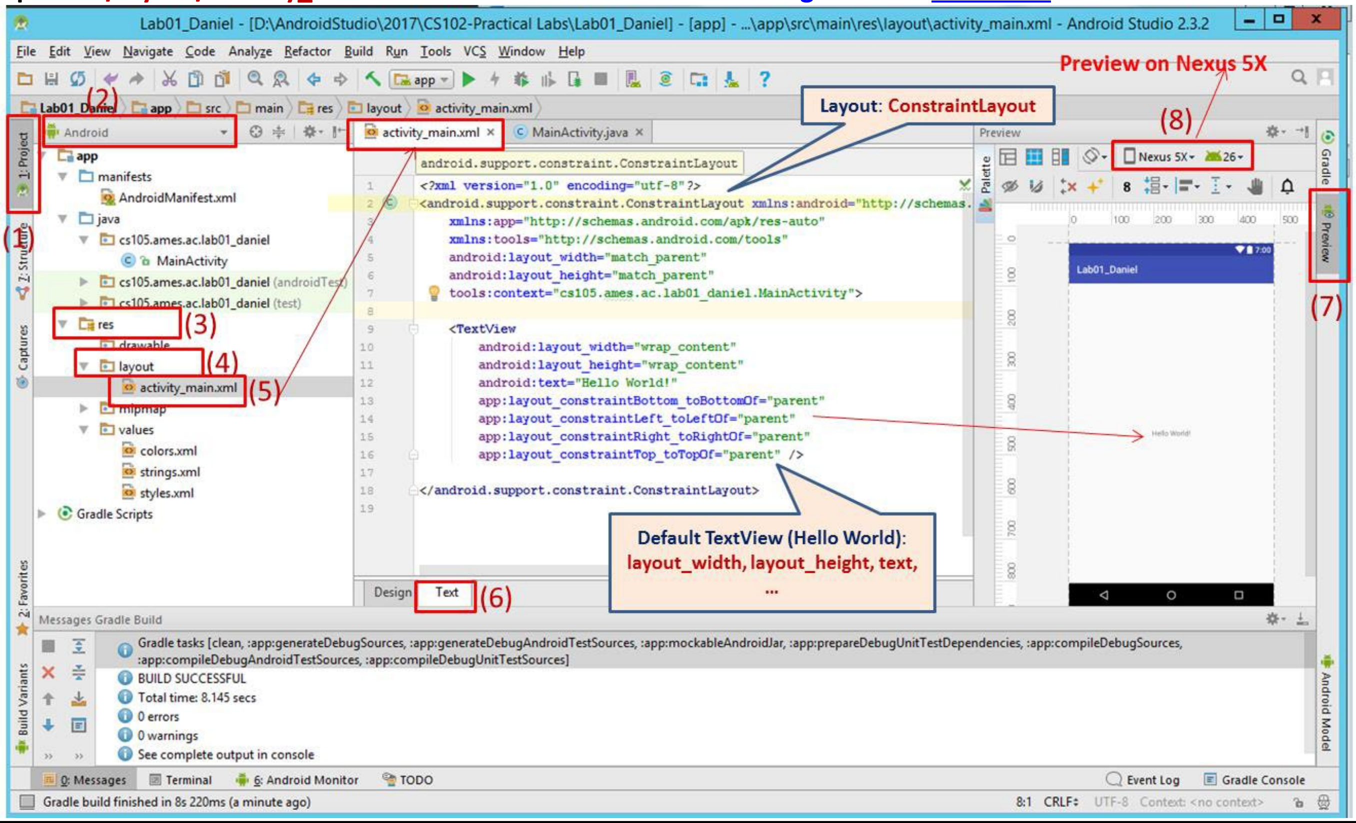
☑ Generate Layout File

Layout Name: activity_main

☑ Backwards Compatibility (AppCompat)

Step 2: Investigate activity_main.xml layout

+ Open res/layout/activity_main.xml file to have a look at the auto-generated XML code:



Lab01_Daniel - [D:\AndroidStudio\2017\CS102-Practical Labs\Lab01_Daniel] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 2.3.2
Preview on Nexus 5X
Layout: ConstraintLayout
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
activity_main.xml
MainActivity.java
Design
Text
Nexus 5X
26
Lab01_Daniel
Hello World!
Default TextView (Hello World): layout_width, layout_height, text, ...

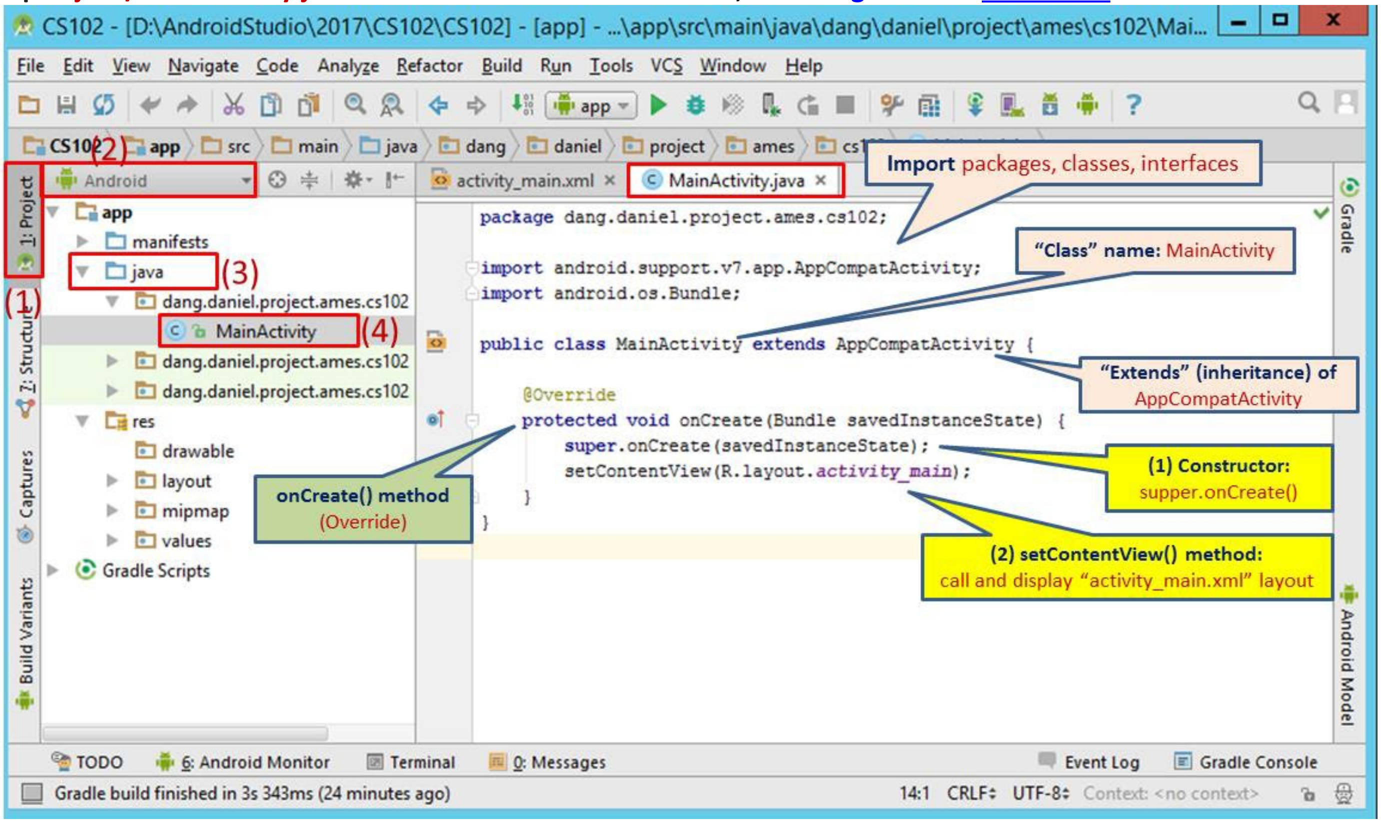
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="cs105.ames.ac.lab01_daniel.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

```
Gradle tasks [clean, :app:generateDebugSources, :app:generateDebugAndroidTestSources, :app:mockableAndroidJar, :app:prepareDebugUnitTestDependencies, :app:compileDebugSources,
:app:compileDebugAndroidTestSources, :app:compileDebugUnitTestSources]
BUILD SUCCESSFUL
Total time: 8.145 secs
0 errors
0 warnings
See complete output in console
```

Gradle build finished in 8s 220ms (a minute ago)

Step 3: Investigate MainActivity.java

+ Open java/MainActivity.java and have a look at its structure, its auto-generated JAVA code:



CS102 - [D:\AndroidStudio\2017\CS102\CS102] - [app] - ...\app\src\main\java\dang\daniel\project\ames\cs102\Mai...
(1)
(2)
(3)
(4)
activity_main.xml
MainActivity.java
Import packages, classes, interfaces
"Class" name: MainActivity
"Extends" (inheritance) of AppCompatActivity
onCreate() method (Override)
(1) Constructor: super.onCreate()
(2) setContentView() method: call and display "activity_main.xml" layout

```java
package dang.daniel.project.ames.cs102;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Gradle build finished in 3s 343ms (24 minutes ago)

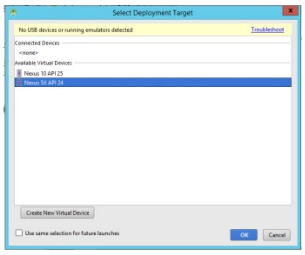
+ Explanation of JAVA code:

- **Import package**, classes, and interfaces;

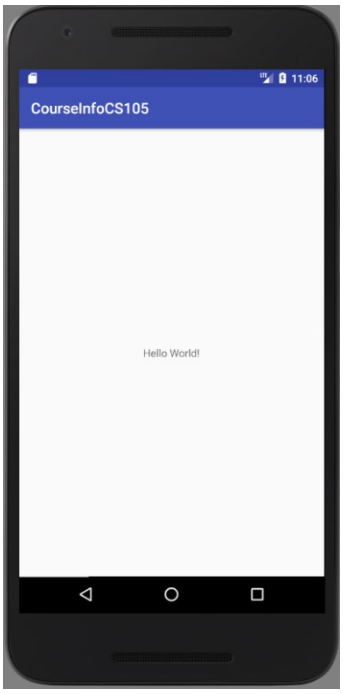
- **Declare "class" name;**
- **Extends** the class of AppCompatActivity parent class;
- **Override the onCreate() method**:
  - Use **super.onCreate() method** as the **constructor**;
  - Use **setContentView() method** to call and display "**activity_main.xml" layout;**

Step 4: Compile & run your app on AVD (Android Virtual Device) for the first time:

+ Click the green triangle button on the tool bar to run your app on Android Virtual Device (AVD).
In this case, we choose Nexus 5X device:


Select Deployment Target
No USB devices or running emulators detected Troubleshoot
Connected Devices
<none>
Available Virtual Devices
Nexus 10 API 25
Nexus 5X API 24
Create New Virtual Device
Use same selection for future launches
OK
Cancel

+ Here we are the outcome:



PART 2: Layout in Android

Layout in Android

Source: https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a **layout** in two ways:

- **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**. The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behaviour. Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems.
- **Instantiate layout elements at runtime**: Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

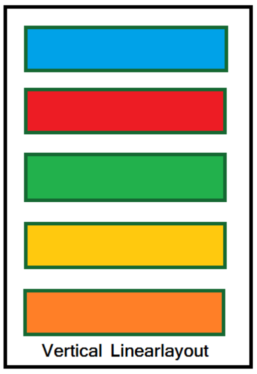
Layout parameters: XML layout attributes named layout_*something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides. All view groups include a width and height (layout_width and layout_height), and each view is required to define them. Many LayoutParams also include optional margins and borders. You can specify width and height with exact measurements, though you probably won't want to do this often. More often, you will use one of these constants to set the width or height:

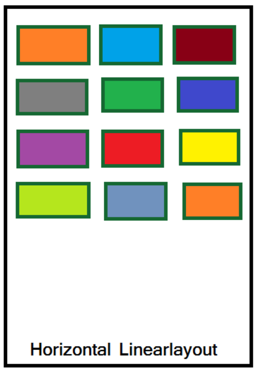
- ***wrap_content*** tells your view to size itself to the dimensions required by its content.
- ***match_parent*** tells your view to become as big as its parent view group will allow.

In general, specifying a layout width and height using absolute units such as pixels is **not recommended**. Instead, using **relative measurements** such as density-independent pixel units (***dp***), ***wrap_content***, or ***match_parent***, is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

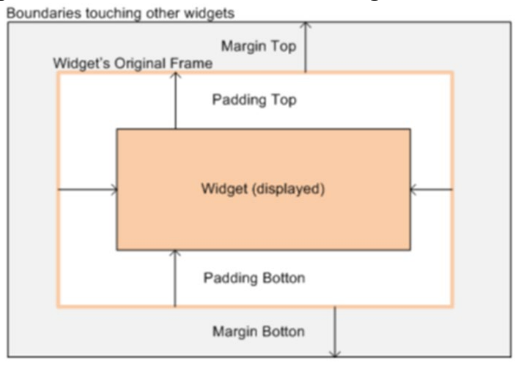
Common Layouts: Below are some of the more common layout types that are built into the Android platform.

Linear Layout

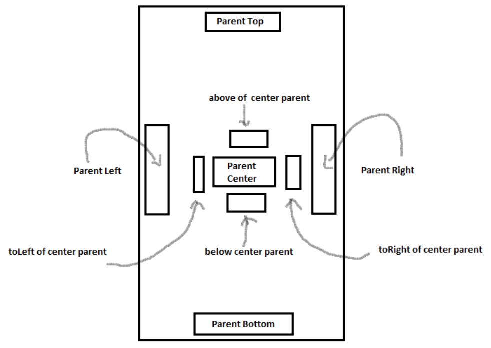




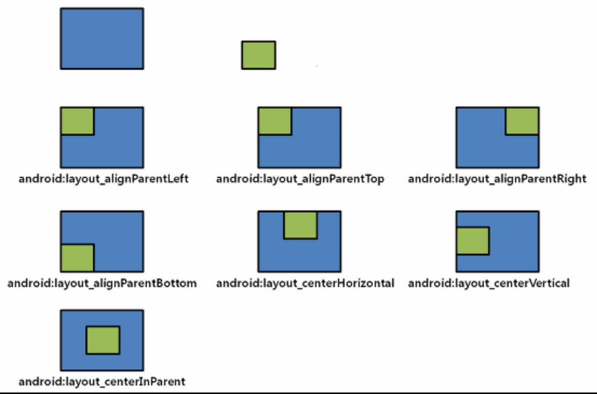
A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.



Relative Layout



Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).


android:layout_alignParentLeft
android:layout_alignParentTop
android:layout_alignParentRight
android:layout_alignParentBottom
android:layout_centerHorizontal
android:layout_centerVertical
android:layout_centerInParent

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the view.
2	**android:layout_width:** This is the width of the layout.
3	**android:layout_height:** This is the height of the layout
4	**android:layout_weight:** This specifies how much of the extra space in the layout should be allocated to the View.
5	**android:layout_x:** This specifies the x-coordinate of the layout.
6	**android:layout_y:** This specifies the y-coordinate of the layout.
7	**android:layout_gravity:** This specifies how child Views are positioned
8	**android:layout_marginTop:** This is the extra space on the top side of the layout.
9	**android:layout_marginBottom:** This is the extra space on the bottom side of the layout.
10	**android:layout_marginLeft:** This is the extra space on the left side of the layout.
11	**android:layout_marginRight:** This is the extra space on the right side of the layout.
13	**android:paddingLeft:** This is the left padding filled for the layout.
14	**android:paddingRight:** This is the right padding filled for the layout.
15	**android:paddingTop:** This is the top padding filled for the layout.
16	**android:paddingBottom:** This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px (Pixels), mm (Millimeters) and finally in (inches).
You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height:

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=match_parent** tells your view to become as big as its parent view.

Step 1: Open activity_main layout and change the "ConstraintLayout" to "RelativeLayout"

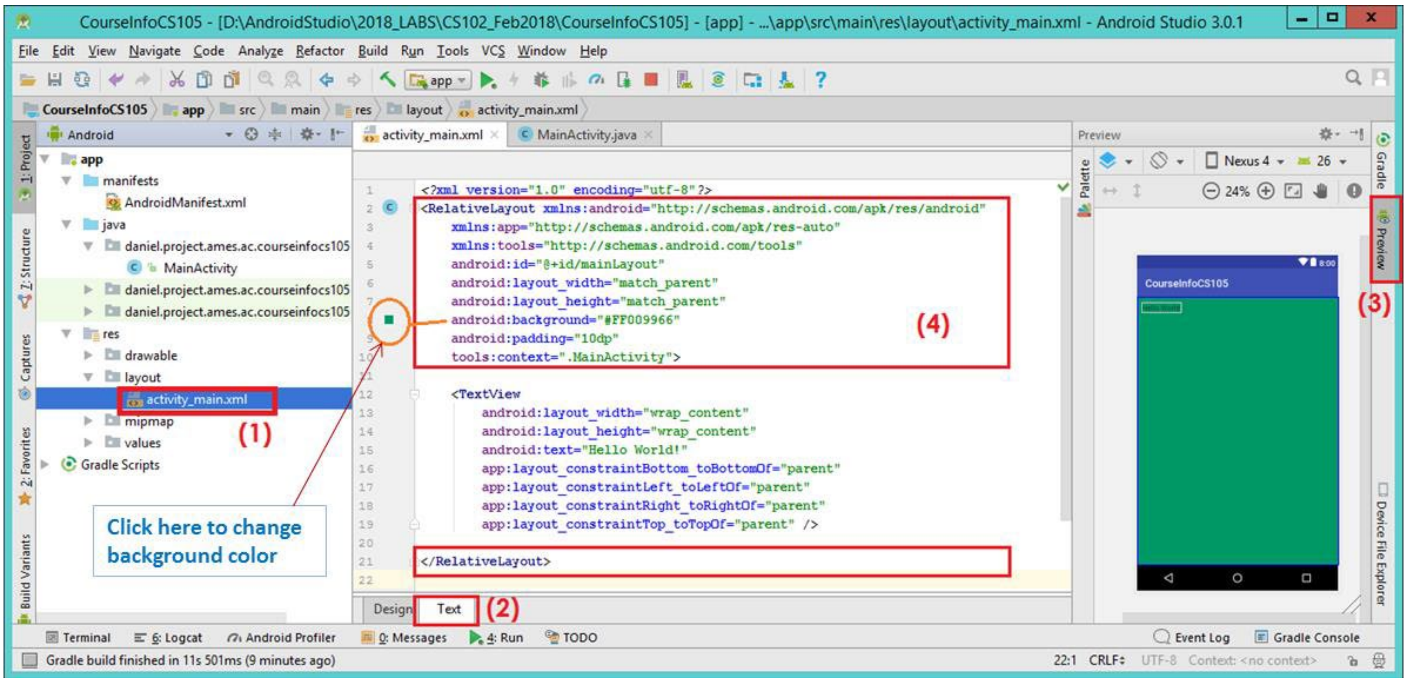
+ Edit the properties of "activity_main.xml" layout as following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF009966"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</RelativeLayout>
```

"background" color

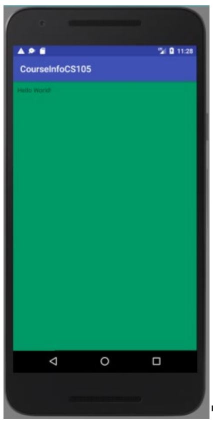
RelativeLayout Properties
+ ID: unique ("mainLayout")
+ layout_width= match_parent
+ layout_height= match_parent
+ padding = 10dp
+ background color= "#FF009966"

+ How it looks like on Android Studio IDE:

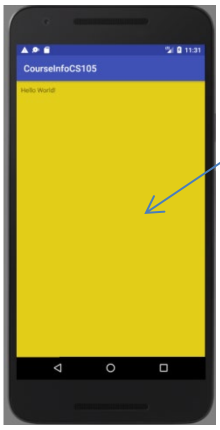

CourseInfoCS105 - [D:\AndroidStudio\2018_LABS\CS102_Feb2018\CourseInfoCS105] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 3.0.1
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CourseInfoCS105 app src main res layout activity_main.xml
Android
app
manifests
AndroidManifest.xml
java
daniel.project.ames.ac.courseinfocs105
MainActivity
daniel.project.ames.ac.courseinfocs105
daniel.project.ames.ac.courseinfocs105
res
drawable
layout
activity_main.xml
mipmap
values
Gradle Scripts
(1)
Click here to change background color
activity_main.xml
MainActivity.java
(4)
Design
Text
(2)
Preview
Nexus 4
26
24%
CourseInfoCS105
(3)
Terminal
6: Logcat
Android Profiler
0: Messages
4: Run
TODO
Event Log
Gradle Console
Gradle build finished in 11s 501ms (9 minutes ago)
22:1 CRLF: UTF-8 Context: <no context>

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#FF009966"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</RelativeLayout>
```

Step 2: Compile & run your app on AVD



➔



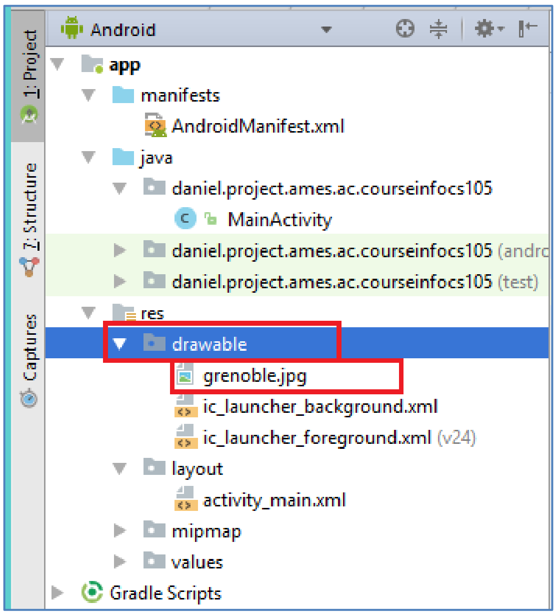
Change background color to yellow:
"Yellow" color code= #e2ce19

Step 3: Change activity_main layout background to an image:

We learn how to use an image as "background" for activity_main. We first copy the background image into "drawable" folder and then assign this image file to layout "background" property.

+ Firstly, copy and paste the image (grenoble.jpg) into the folder "drawable":

➔



```
Android
▼ app
    ▼ manifests
          AndroidManifest.xml
    ▼ java
        ▼ daniel.project.ames.ac.courseinfocs105
              MainActivity
        ▶ daniel.project.ames.ac.courseinfocs105 (andrc
        ▶ daniel.project.ames.ac.courseinfocs105 (test)
    ▼ res
        ▼ drawable
              grenoble.jpg
              ic_launcher_background.xml
              ic_launcher_foreground.xml (v24)
        ▼ layout
              activity_main.xml
        ▶ mipmap
        ▶ values
▶ Gradle Scripts
```

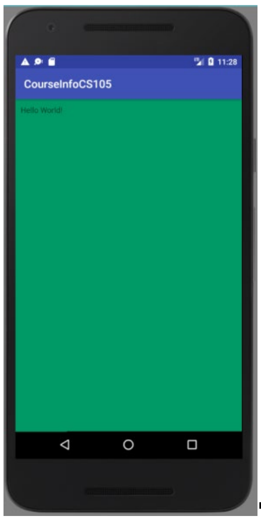
Image: grenoble.jpg and it's copied and pasted in "drawable" folder

+ Secondly, open activity_main.xml and assign the file grenoble.jpg to "background" property:

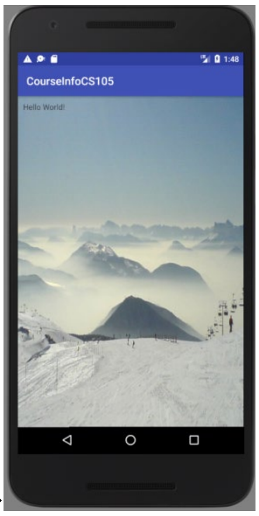
```
android:background="@drawable/grenoble"
```

Change Background to an image

+ Run your app on AVD to see the result



➔

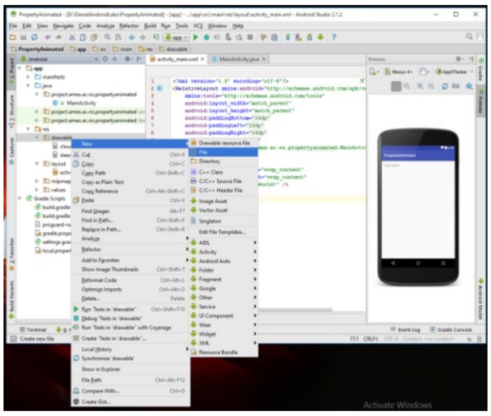


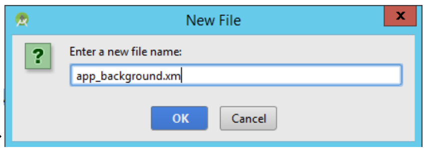
Step 4: Change activity_main layout background to a gradient color

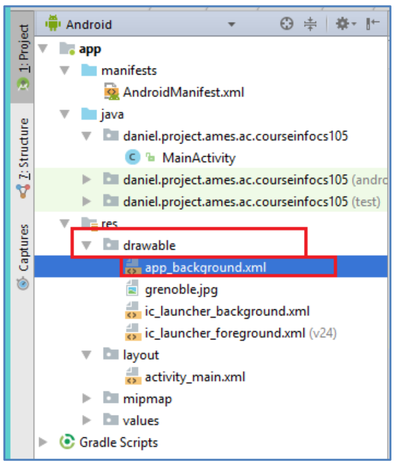
Now we learn how to define gradient color as background color for activity_main. We have to design our own gradient color in XML file and then assign this xml file to layout "background" property.

+ Firstly, add a new XML file to the project:

*_Right click on "**drawable**" folder → "**New**" → "**File**" → a window is appeared*
*_Enter the "File name": **app_background.xml***
_Click "Ok"







Android
app
manifests
AndroidManifest.xml
java
daniel.project.ames.ac.courseinfocs105
MainActivity
daniel.project.ames.ac.courseinfocs105 (andro
daniel.project.ames.ac.courseinfocs105 (test)
res
drawable
app_background.xml
grenoble.jpg
ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
layout
activity_main.xml
mipmap
values
Gradle Scripts
1: Project
7: Structure
Captures

Create a **New File** (XML) named "app_background.xml";

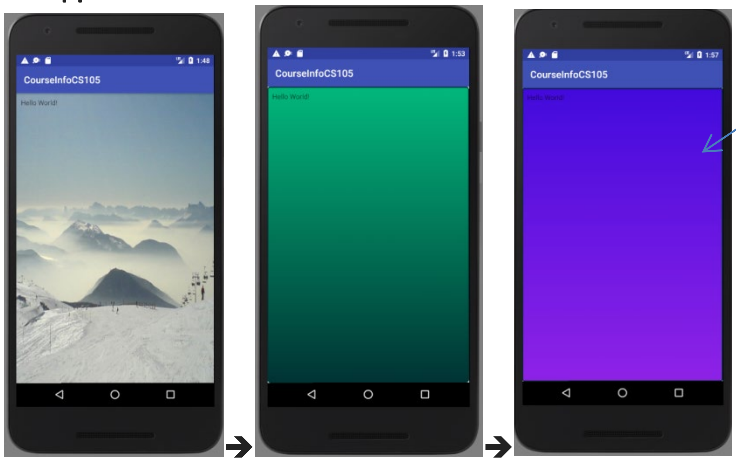
+ Secondly, open the *app_background.xml* (inside "*drawable*" folder) and edit it as below:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:dither="true"
    android:shape="rectangle">
    <!--Define gradient colour-->
    <gradient
        android:angle="90"
        android:endColor="#FF02B77B"
        android:startColor="#FF003333" />
    <!--Define corner-->
    <corners android:radius="5dp" />
    <!--Define stroke-->
    <stroke
        android:width="2dp"
        android:color="#FF003333" />
</shape>
```

+ Thirdly, open activity_main.xml file and assign app_background.xml to "background" property in activity_main.xml layout as following:

```
android:background="@drawable/app_background"
```

+ Run your app on AVD to see the result:

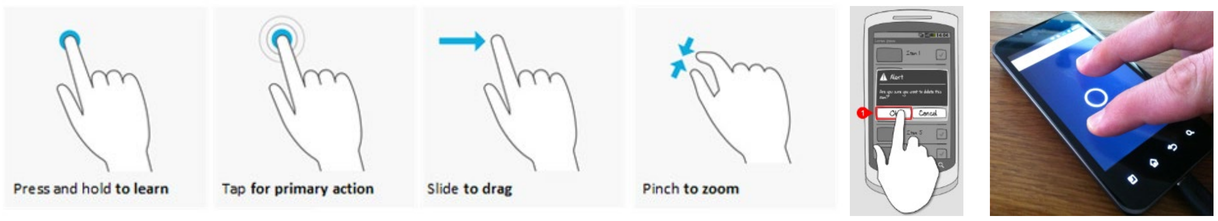


Change background to gradient color:
android:endColor="#420adc"
android:startColor="#8e22e6"

Step 5: Set "Click listener" for layout to interact with users: When users touch screen, the app pops up a message "Welcome to CS102 paper".

User Interaction Handling in Android

Source: https://www.tutorialspoint.com/android/android_event_handling.htm



User events: press, click, touch, tap, slide, pinch, etc.

Events are a useful way to collect data about a **user's interaction** with interactive components of Applications. Like **button presses** or **screen touch** etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are following three concepts related to Android Event Management:

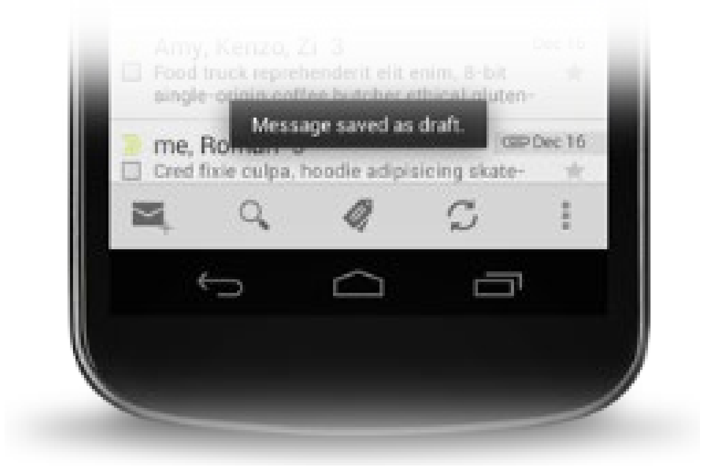
- **Event Listeners** – An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners Registration** – Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** – When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

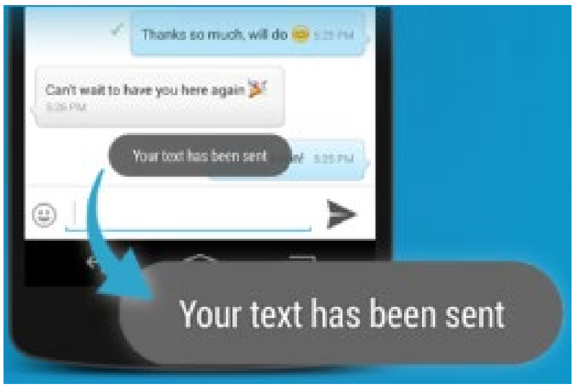
Event Listeners & Event Handlers:

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use **onClick() event handler** to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This is called when the context menu is being built(as the result of a sustained "long click)

Notification message in Android:

In Android, Toast is a **notification message** that pop up, display a certain amount of time, and automatically fades in and out, most people just use it for **debugging** or **notification purpose**.


Amy, Kenzo, Zi 3
Food truck reprehenderit elit enim, 8-bit single-origin coffee butcher ethical gluten-
Message saved as draft.
me, Roman 3
Dec 16
Cred fixie culpa, hoodie adipisicing skate-



In this part, when users touch screen, in fact they touch RelativeLyaout, the app will pop up a message "Welcome to CS102 paper".

+ Open to MainActivity.java file, keep the auto-generated codes.

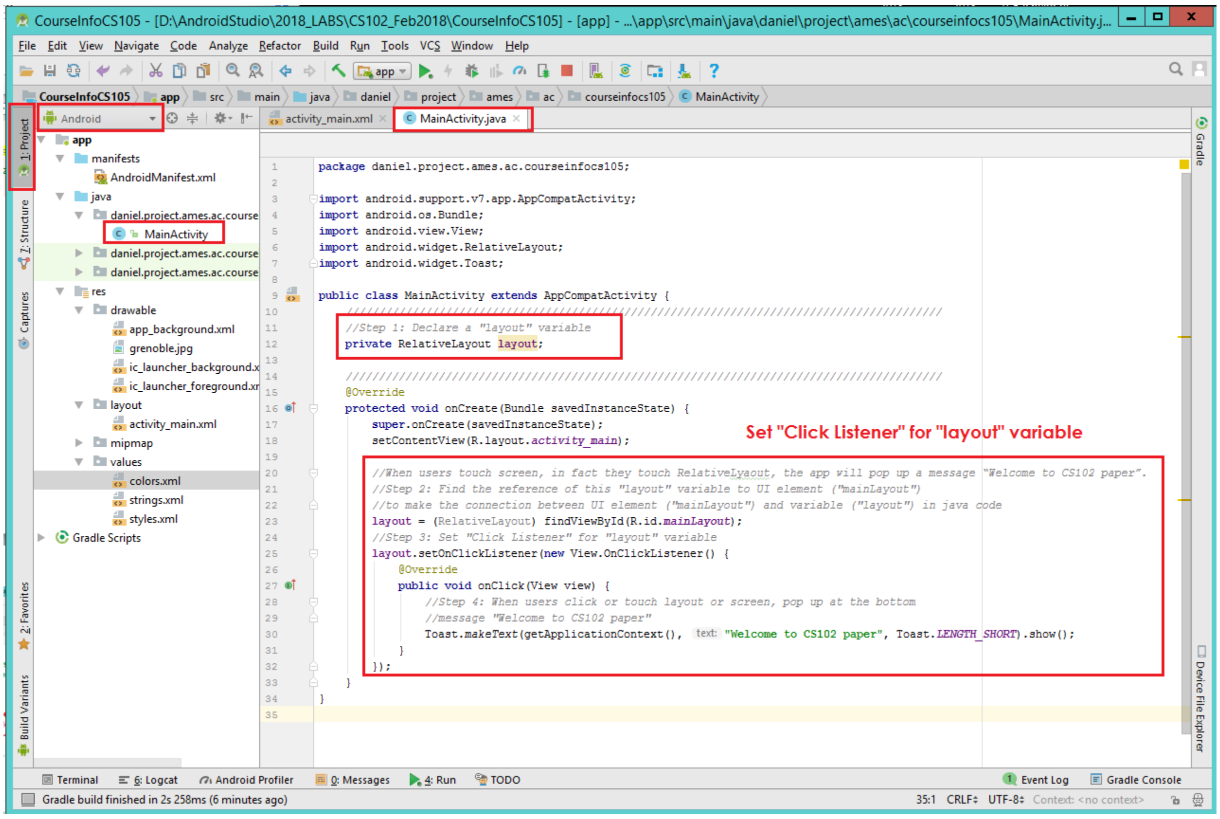
_Right below the class declaration, declare a variable "layout" from Relativelayout class:

```
///////////////////////////////////////////////////////////////////////////////////////////
//Step 1: Declare a "layout" variable
private RelativeLayout layout;
```

_Inside onCreate() method, add the below java code:

```
//When users touch screen, in fact they touch RelativeLyaout,
// the app will pop up a message "Welcome to CS102 paper".
//Step 2: Find the reference of this "layout" variable to UI element ("mainLayout")
//to make the connection between UI element ("mainLayout") and variable ("layout") in java code
layout = (RelativeLayout) findViewById(R.id.mainLayout);
//Step 3: Set "Click Listener" for "layout" variable
layout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Step 4: When users click or touch layout or screen, pop up at the bottom
        //message "Welcome to CS102 paper"
        Toast.makeText(getApplicationContext(), "Welcome to CS102 paper", Toast.LENGTH_SHORT).show();
    }
});
```

+ The MainActivity.java file now looks like:

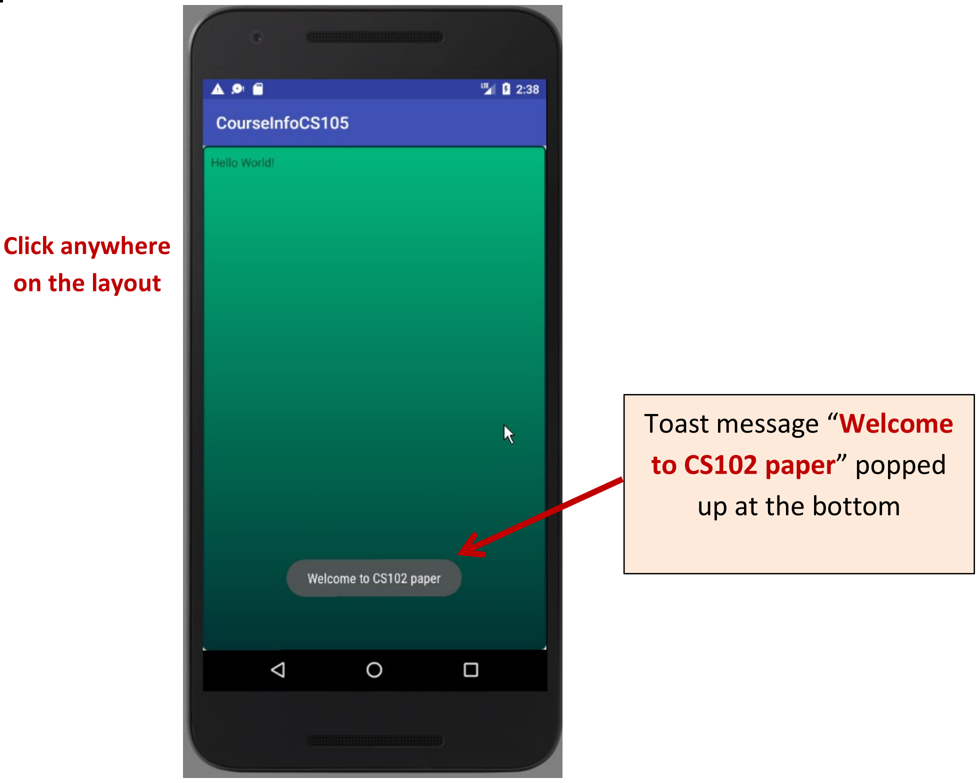


```
package daniel.project.ames.ac.courseinfocs105;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.RelativeLayout;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    ///////////////////////////////////////////////////////////////////////////////////////////
    //Step 1: Declare a "layout" variable
    private RelativeLayout layout;

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

        //When users touch screen, in fact they touch RelativeLyaout, the app will pop up a message "Welcome to CS102 paper".
        //Step 2: Find the reference of this "layout" variable to UI element ("mainLayout")
        //to make the connection between UI element ("mainLayout") and variable ("layout") in java code
        layout = (RelativeLayout) findViewById(R.id.mainLayout);
        //Step 3: Set "Click Listener" for "layout" variable
        layout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Step 4: When users click or touch layout or screen, pop up at the bottom
                //message "Welcome to CS102 paper"
                Toast.makeText(getApplicationContext(), text: "Welcome to CS102 paper", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Set "Click Listener" for "layout" variable

CourseInfoCS105 - [D:\AndroidStudio\2018_LABS\CS102_Feb2018\CourseInfoCS105] - [app] - ...\app\src\main\java\daniel\project\ames\ac\courseinfocs105\MainActivity.j...
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
CourseInfoCS105 app src main java daniel project ames ac courseinfocs105 MainActivity
Android
activity_main.xml
MainActivity.java
app
manifests
AndroidManifest.xml
java
daniel.project.ames.ac.course
MainActivity
res
drawable
app_background.xml
grenoble.jpg
ic_launcher_background.x
ic_launcher_foreground.xr
layout
activity_main.xml
mipmap
values
colors.xml
strings.xml
styles.xml
Gradle Scripts
Terminal
6: Logcat
Android Profiler
0: Messages
4: Run
TODO
Event Log
Gradle Console
Gradle build finished in 2s 258ms (6 minutes ago)
35:1 CRLF UTF-8 Context: <no context>

+ Compile and run your app on AVD to see the result:



2:38
CourseInfoCS105
Hello World!
Welcome to CS102 paper
Click anywhere on the layout
Toast message "Welcome to CS102 paper" popped up at the bottom

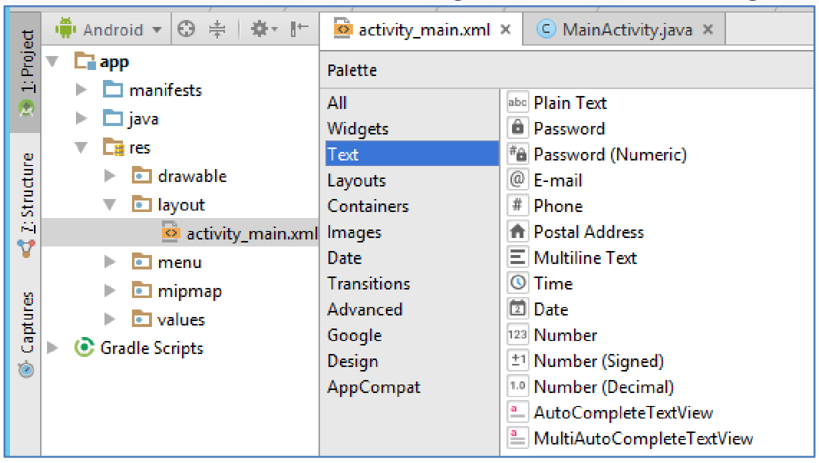
PART 3: ADD TEXTVIEW

A SUBCLASS OF “VIEW” SUPERCLASS

TextView

Source: https://www.tutorialspoint.com/android/android_textview_control.htm

A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor however the basic class is configured to not allow editing.



+ dp:	Density-independent Pixels
+ sp:	Scale-independent Pixels
+ pt:	Points which is 1/72 of an inch
+ px:	Pixels
+ mm:	Millimeters
+ in:	inches

TextView Attributes

Following are the important attributes related to **TextView control**. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the control.
2	**android:layout_width:** specify width with exact measurements **android:layout_heigth:** specify height with exact measurements • ***wrap_content***: tells your view to size itself to the dimensions required by its content. • ***match_parent***: tells your view to become as big as its parent view group will allow.
3	**android:gravity:** Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
4	**android:text:** Text to display
5	**android:textColor:** Definet the text color
6	**android:textSize:** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).
7	**android:textStyle:** Style (bold, italic, bold, italic) for the text. You can use or more of the following values separated by '|'.
8	**android:inputType:** The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.

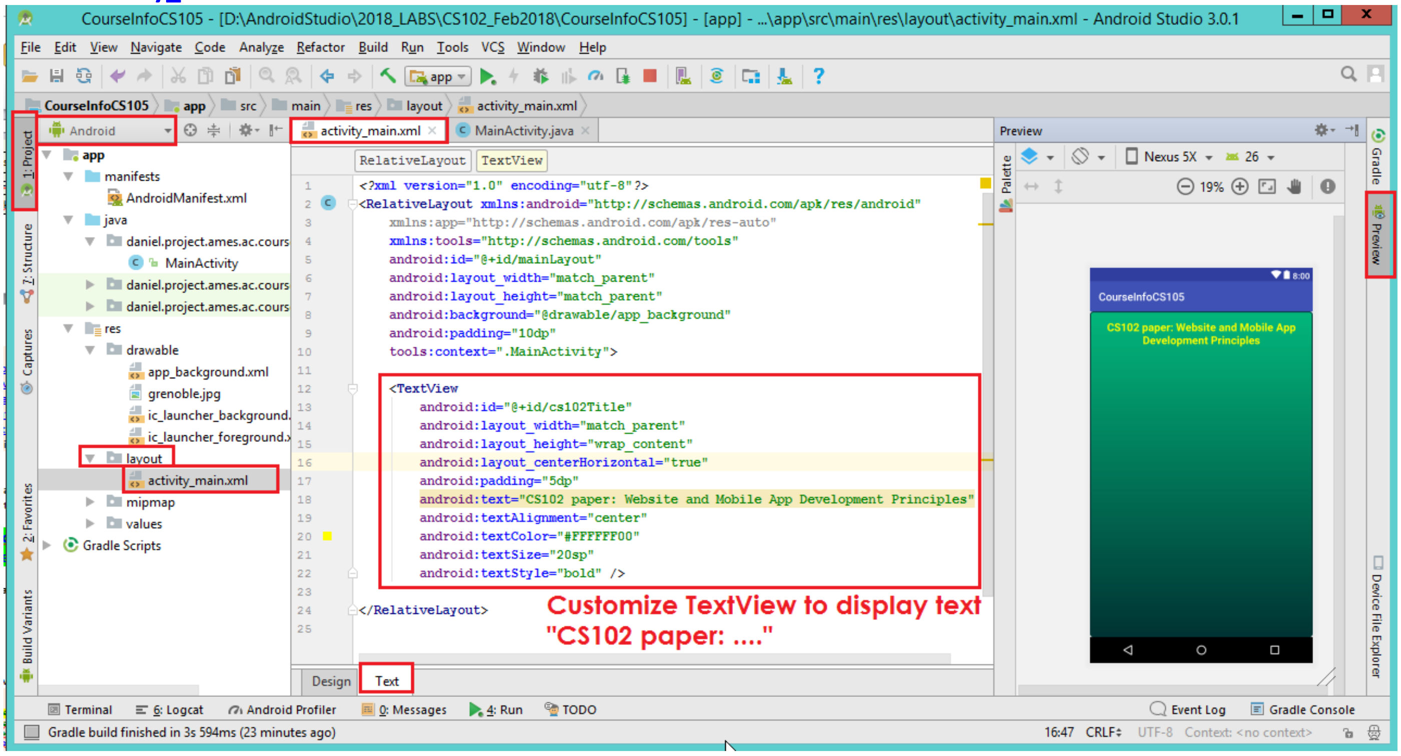
Step 1: Change the TextView to display text "CS102 paper: Website and Mobile App Development Principles"

So far, you see an **auto-generated TextView element** inside the RelativeLayout displaying "Hello world". Now we will learn how to customize the TextView in Android to our application.

+ Open activity_main.xml file and modify TextView element the as below:

```
<TextView
    android:id="@+id/cs102Title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:padding="5dp"
    android:text="CS102 paper: Website and Mobile App Development Principles"
    android:textAlignment="center"
    android:textColor="#FFFFFF00"
    android:textSize="20sp"
    android:textStyle="bold" />
```

+ The activity_main.xml looks like:



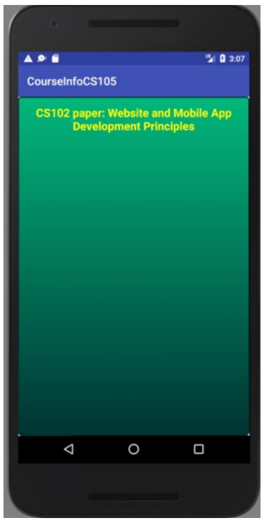
CourseInfoCS105 - [D:\AndroidStudio\2018_LABS\CS102_Feb2018\CourseInfoCS105] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 3.0.1

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_background"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/cs102Title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:padding="5dp"
        android:text="CS102 paper: Website and Mobile App Development Principles"
        android:textAlignment="center"
        android:textColor="#FFFFFF00"
        android:textSize="20sp"
        android:textStyle="bold" />

</RelativeLayout>
```

Customize TextView to display text "CS102 paper: ...."

Gradle build finished in 3s 594ms (23 minutes ago)

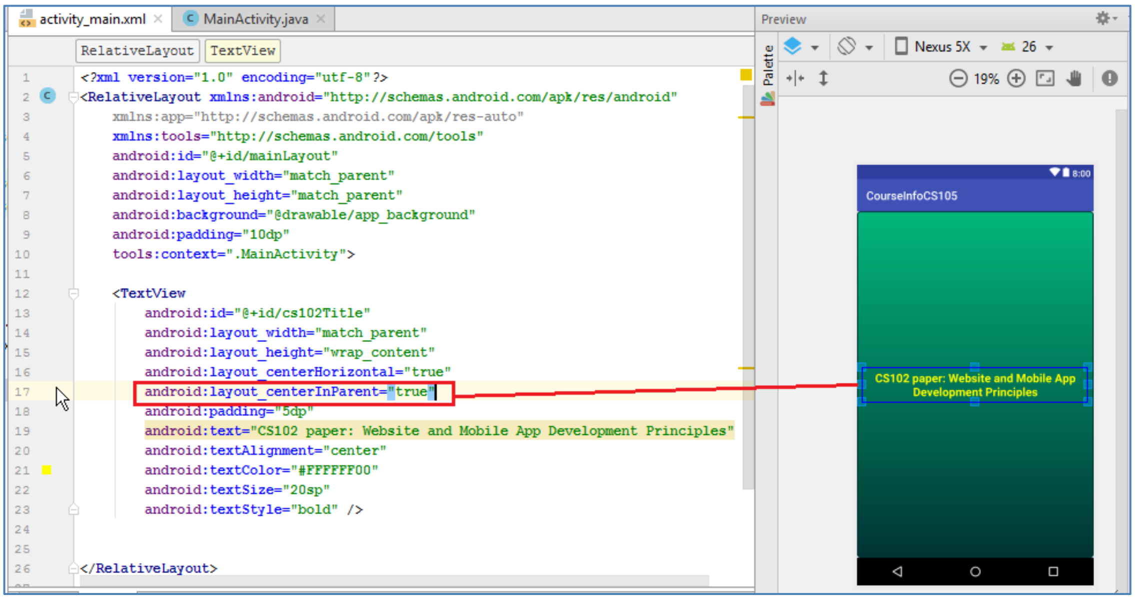
+ Compile and run your app on AVD to see the result:



Step 2: Move the TextView to the centre of screen or layout, modify textColor & textSize, and format the “text” property

+ Add one more property “layout_centerInParent” to TextView so that it’s located in the centre of screen:

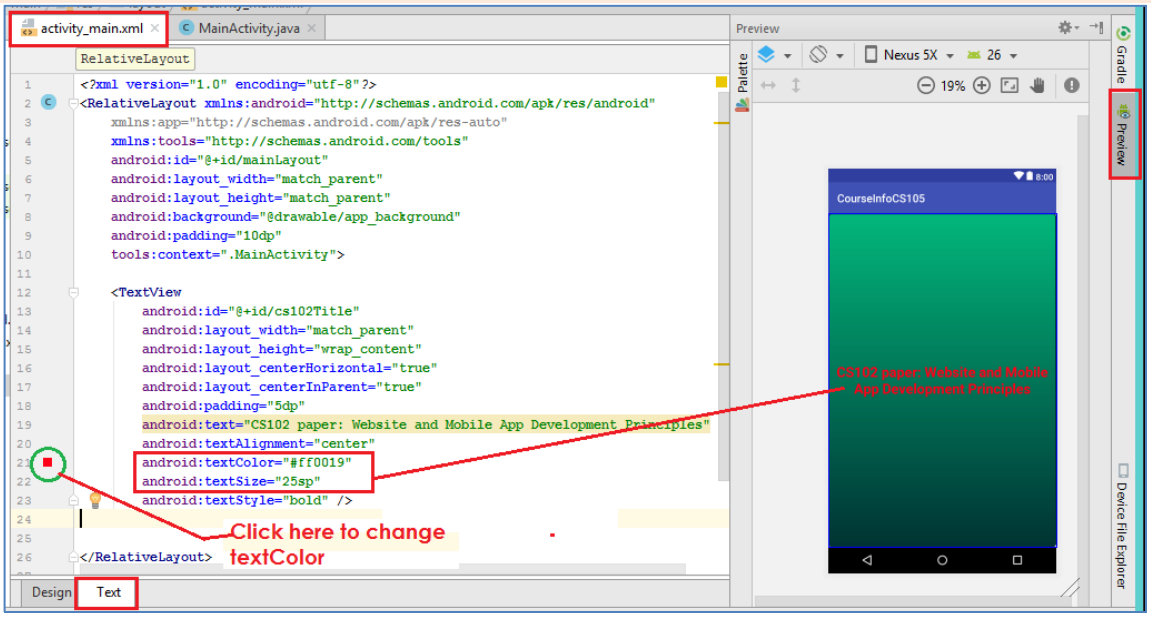
```
android:layout_centerInParent="true"
```



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_background"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/cs102Title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerInParent="true"
        android:padding="5dp"
        android:text="CS102 paper: Website and Mobile App Development Principles"
        android:textAlignment="center"
        android:textColor="#FFFFFF00"
        android:textSize="20sp"
        android:textStyle="bold" />


</RelativeLayout>
```

+ Change the textColor to RED & make textSize bigger (=25sp):

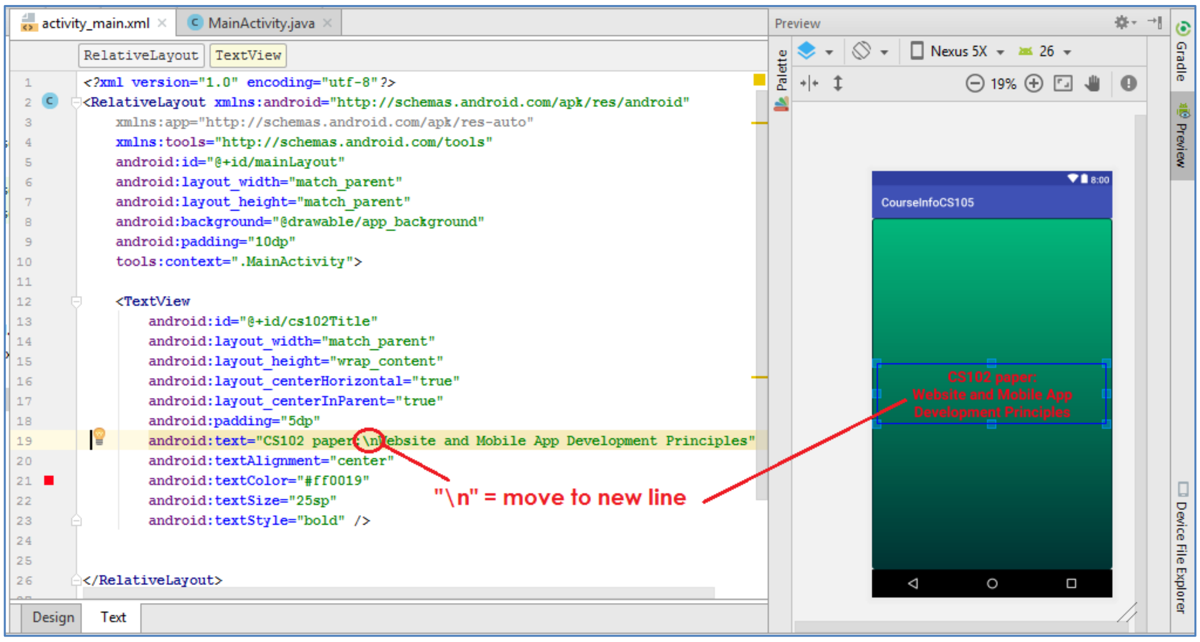
```
android:textColor="#ff0019"
android:textSize="25sp"
```



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_background"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/cs102Title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerInParent="true"
        android:padding="5dp"
        android:text="CS102 paper: Website and Mobile App Development Principles"
        android:textAlignment="center"
        android:textColor="#ff0019"
        android:textSize="25sp"
        android:textStyle="bold" />


</RelativeLayout>
```

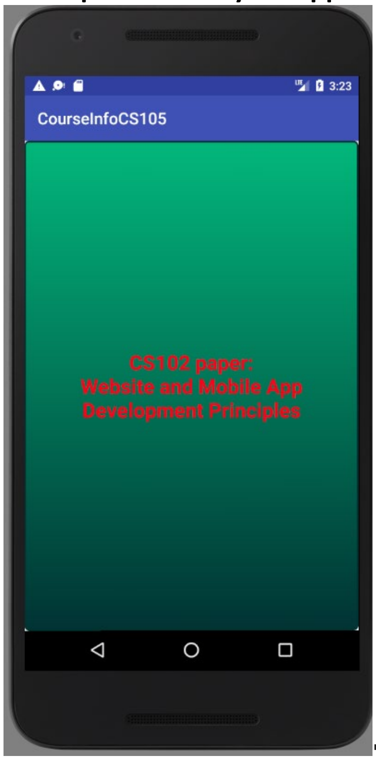
+ Now, We want to display long text “Paper CS102: Website and Mobile App Development Principles” into 2 lines: first line contains only “Paper CS102:” and the second line is “Website and Mobile App Development Principles”. In order to do that, it’s simple to add “\n” where we want to move to new line:

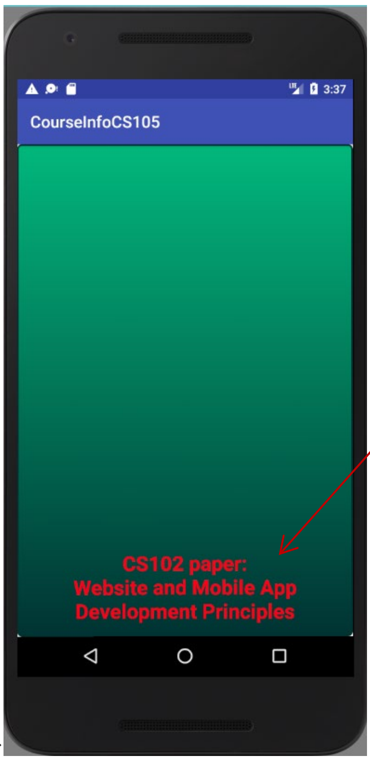
```
android:text="CS102 paper:\nWebsite and Mobile App Development Principles"
```



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_background"
    android:padding="10dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/cs102Title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerInParent="true"
        android:padding="5dp"
        android:text="CS102 paper:\nWebsite and Mobile App Development Principles"
        android:textAlignment="center"
        android:textColor="#ff0019"
        android:textSize="25sp"
        android:textStyle="bold" />


</RelativeLayout>
```

+ Compile and run your app on AVD to see the result:





Move TextView to the bottom of screen:
`android:layout_alignParentBottom="true"`

There is no way to underline a text in layout, such as CS102 paper: We have to format texts (underline for example) in **strings.xml file**. You will learn this in next part.

Step 4: Create "color" variables in colors.xml and "text" variables in strings.xml

Android Resources Organizing & Accessing for your project

Source: https://www.tutorialspoint.com/android/android_resources.htm

There are many more items which you use to build a **good Android application**. Apart from coding (in java) for the application, you take care of **various other resources** like **static content** that your code uses, such as **bitmaps**, **colors**, **layout definitions**, **user interface strings**, **animation instructions**, and more. These resources are always maintained separately in various sub-directories under **res/ directory** of the project.

Organize sources in Android Studio:

Directory	Resource Type
anim/	XML files that define property animations. They are saved in **res/anim/ folder** and accessed from the **R.anim** class.
color/	XML files that define a state list of colors. They are saved in **res/color/** and accessed from the **R.color** class.
drawable/	Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in **res/drawable/** and accessed from the **R.drawable** class.
layout/	XML files that define a user interface layout. They are saved in **res/layout/** and accessed from the **R.layout** class.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in **res/menu/** and accessed from the **R.menu** class.
raw/	Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files.
values/	XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory: • **arrays.xml** for resource arrays, and accessed from the **R.array** class. • **integers.xml** for resource integers, and accessed from the **R.integer** class. • **bools.xml** for resource boolean, and accessed from the **R.bool** class. • **colors.xml** for color values, and accessed from the **R.color** class. • **dimens.xml** for dimension values, and accessed from the **R.dimen** class. • **strings.xml** for string values, and accessed from the **R.string** class. • **styles.xml** for styles, and accessed from the **R.style** class.
xml/	Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

+ Firstly, open the colors.xml file in "values" folder.
Keep all the content and add the "text_color" variable below existing code that defines the textColor of TextView:

colors.xml:
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <!--Add text_color variable to define text color of TextView-->
    <color name="text_color">#ff0019</color>
</resources>
```

+ Secondly, open the strings.xml file in "values" folder.
Keep all the content and add the "cs102_title" variable below existing code that is content of the "text" of TextView:

strings.xml:
```
<resources>
    <string name="app_name">CourseInfoCS102</string>
```

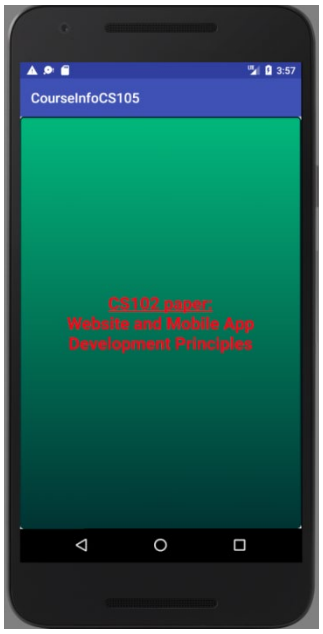
```xml
    <!--Add "cs102_title" variable containing "text" of TextView -->
    <string name="cs102_title"><u>CS102 paper:\n</u>Website and Mobile App Development Principles</string>
</resources>
```

We also format the first line **"CS102 paper:"** to have underlined by using **<u> </u> tag**

+ Finally, open the activity_main.xml layout file and assign these two variables (text_color & cs102_title) to 2 properties "textColor" and "text" of TextView as below:

```xml
<TextView
    android:id="@+id/cs102Title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:padding="5dp"
    android:text="@string/cs102_title"
    android:textAlignment="center"
    android:textColor="@color/text_color"
    android:textSize="25sp"
    android:textStyle="bold" />
```

+ Compile and run your app on AVD to see the result:



CS102 paper:
Website and Mobile App
Development Principles

+ Similarly, you can add **"dimension" variables** in **dimens.xml file**.

Step 5: Set "Click listener" for TextView to interact with users
When users touch the TextView, TextView will change to WHITE color, the 'text" content is changed to "Have a good day!", and is moved to the bottom of screen

+ Open to MainActivity.java, keep the existing codes:

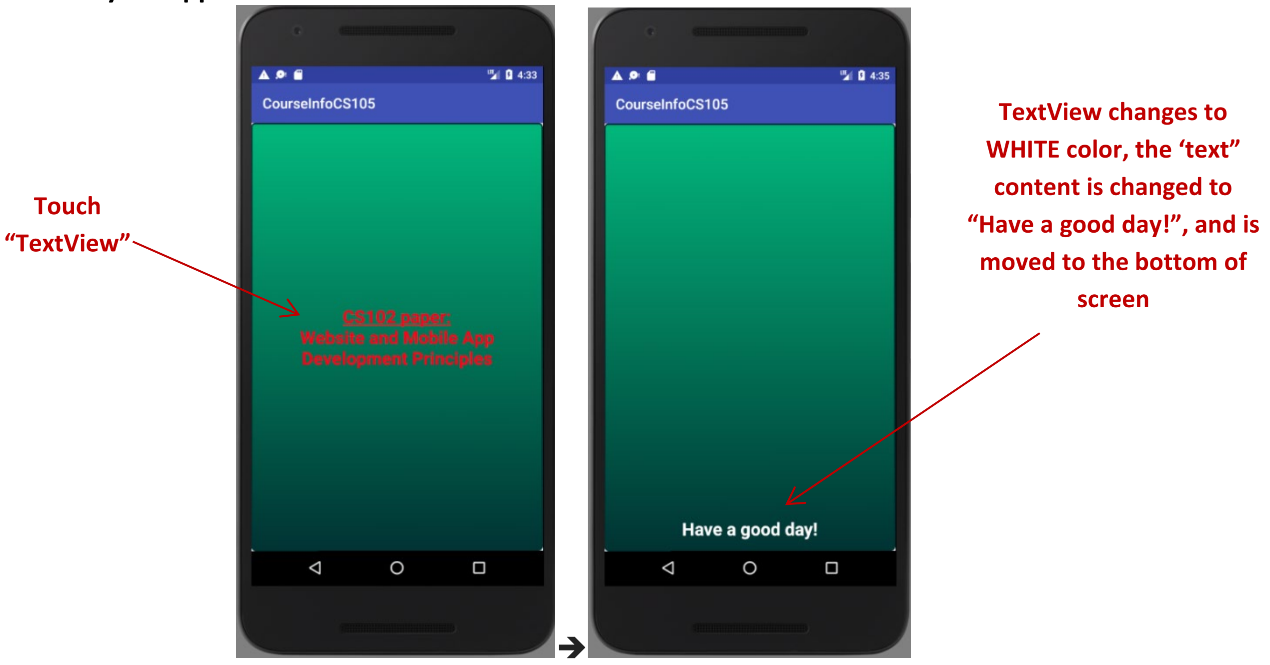
_Right below the class declaration, declare a variable "courseTitle" from TextView class:

```
//Step 1: Declare a "courseTitle" variable
private TextView courseTitle;
```

_Inside onCreate() method, add the below java code:

```
//////////////////////////////////////////////////////////////////////////////
//When users touch TextView, TextView will change to WHITE color and move to the bottom of screen
//Step 2: Find the reference of the "courseTitle" variable to UI element ("cs102Title")
//to make the connection between UI element ("cs102Title") and variable ("courseTitle") in java code
courseTitle = (TextView) findViewById(R.id.cs102Title);
//Step 3: Set "Click Listener" for "courseTitle" variable
courseTitle.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Step 4: When users click TextView has been detected
        //Change the text color to WHITE by calling setTextColor() method
        courseTitle.setTextColor(Color.WHITE);
        //Change the "text" content to "Have a good day!" by calling setText() method
        courseTitle.setText("Have a good day!");
        //Move the TextView to the bottom of screen by addRule RelativeLayout.ALIGN_PARENT_BOTTOM
        RelativeLayout.LayoutParams params = new
RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
           RelativeLayout.LayoutParams.WRAP_CONTENT);
        //Add rule: align "TextView" to the bottom of screen
        params.addRule(RelativeLayout.ALIGN_PARENT_BOTTOM, RelativeLayout.TRUE);
        //Set above rules to "TextView"
        courseTitle.setLayoutParams(params);
    }
});
```

+ Compile and run your app on AVD to see the result:



In order to align TextView to centre horizontal, add the below rule:

```
//Set rule: align "TextView" to center horizontal
params.addRule(RelativeLayout.CENTER_HORIZONTAL);
```

EXPERIMENT (OPTIONAL):
Add java code so that when click the TextView (Have a good day), restore its content "CS102 paper: Website and Mobile App Development Principles" in RED color and appears in the center of screen.

PART 4: CHANGE APP-STYLE AND APP-THEME

Android - Styles and Themes

Source: https://www.tutorialspoint.com/android/android_styles_and_themes.htm

A **style** resource defines the format and look for a UI. A style can be applied to an individual View (from within a layout file) or to an entire Activity or application (from within the manifest file).

Define Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under **res/values/** directory of your project and will have **<resources>** as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the **.xml extension**.

You can **define multiple styles per file using <style> tag** but each style will have its name that uniquely identifies the style. Android style attributes are set using **<item>** tag as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:capitalize">characters</item>
    <item name="android:typeface">monospace</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textColor">#00FF00</item>/>
  </style>
</resources>
     <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>
```

Default Styles & Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the **R.style** class.
To use the styles listed here, **replace all underscores in the style name with a period**. For example, you can apply the **Theme_NoTitleBar** theme with **"@android:style/Theme.NoTitleBar"**.

You can see the following source code for Android styles and themes:

- Android Styles (styles.xml)
- Android Themes (themes.xml)

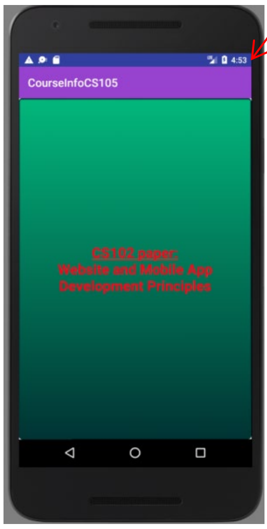
Step 1: Change app style

+ Open /res/values/colors.xml file and change the colorPrimary from "#3F51B5" to "#9742CF":

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#9742CF</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <!--Add text_color variable to define text color of TextView-->
    <color name="text_color">#ff0019</color>
</resources>
```

Change the color of "action bar" to purple color

+ Compile and run your app on AVD to see the result:

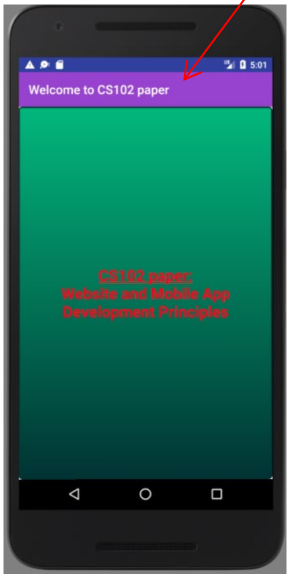


Step 2: Change app_name

+ Open the *res/values/strings.xml* file, modify the app_name to "Welcome to CS102 paper":

```
<resources>
    <string name="app_name">Welcome to CS102 paper</string>

    <!--Add "cs102_title" variable containing "text" of TextView -->
    <string name="cs102_title"><u>CS102 paper:\n</u>Website and Mobile App Development Principles</string>
</resources>
```

+ Run your app on AVD to see the result:



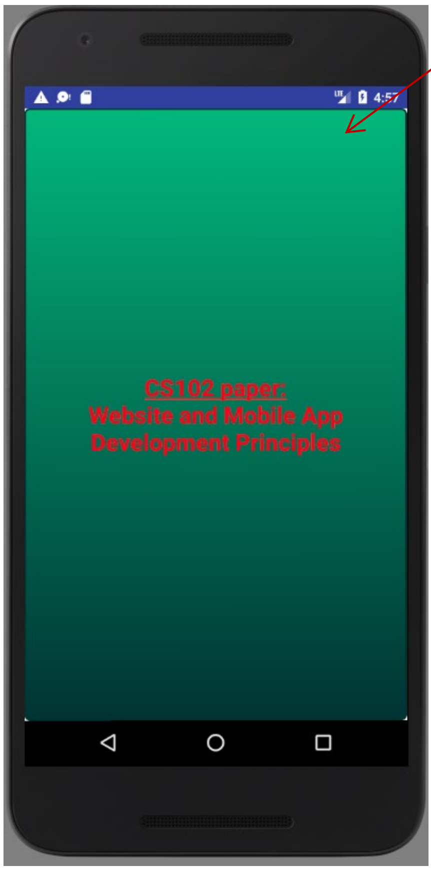
Step 3: Change app theme

+ Open /res/values/styles.xml file and change the "AppTheme" from "Theme.AppCompat.Light.DarkActionBar" to "Theme.AppCompat.Light.NoActionBar"

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

+ Run your app on AVD to see the result:

No "Action" Bar at all



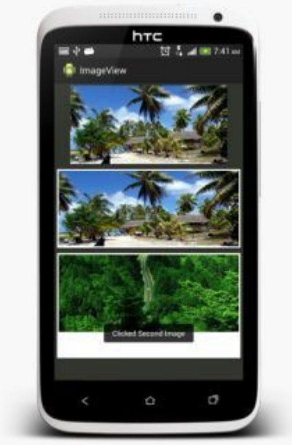
PART 5: ADD IMAGEVIEW

A SUBCLASS OF "VIEW" SUPERCLASS

ImageView is to displays image resources, for example Bitmap or Drawable resources. ImageView is also commonly used to apply tints to an image and handle image scaling.

The following XML snippet is a common example of using an ImageView to display an image resource:

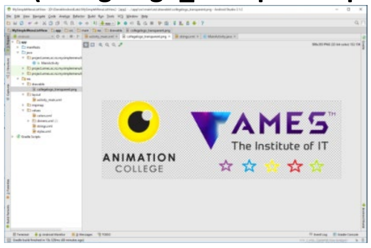
```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@mipmap/ic_launcher"
        />
</LinearLayout>
```



Step 1: Add an ImageView on layout to display the college logo

In this part, we will add an ImageView to display our college logo (collegelogo_transparent.png) on screen.

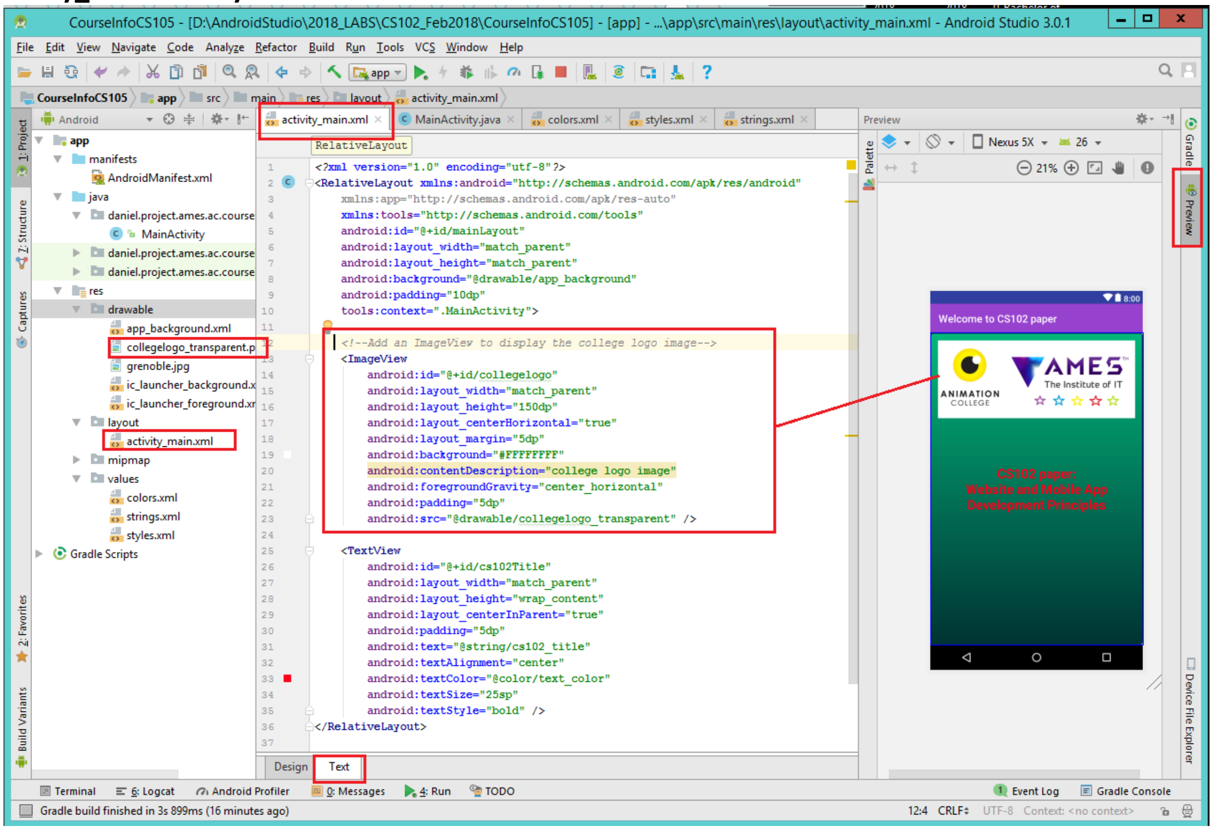
+ Copy and paste the college logo image file (collegelogo_transparent.png) to the folder "drawable"



+ Open activity_main.xml file and add an ImageView to your layout, above the TextView element, assign its image source ("src" property) to collegelogo_transparent.png image as below:

```
<!--Add an ImageView to display the college logo image-->
<ImageView
    android:id="@+id/collegelogo"
    android:layout_width="match_parent"
    android:layout_height="150dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:layout_margin="5dp"
    android:background="#FFFFFFFF"
    android:contentDescription="college logo image"
    android:foregroundGravity="center_horizontal"
    android:padding="5dp"
    android:scaleType="fitCenter"
    android:src="@drawable/collegelogo_transparent" />
```

+ The activity_main.xml layout file looks like:

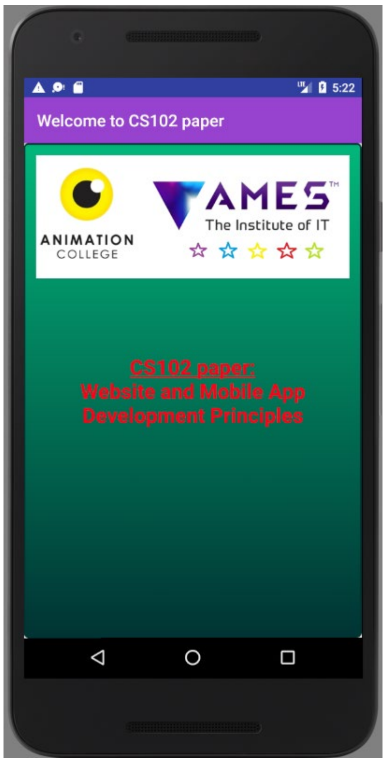


CourseInfoCS105 - [D:\AndroidStudio\2018_LABS\CS102_Feb2018\CourseInfoCS105] - [app] - ...\app\src\main\res\layout\activity_main.xml - Android Studio 3.0.1

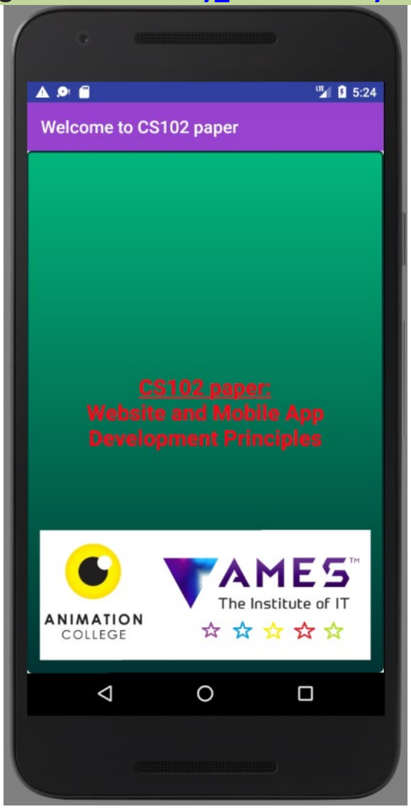
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_background"
    android:padding="10dp"
    tools:context=".MainActivity">

    <!--Add an ImageView to display the college logo image-->
    <ImageView
        android:id="@+id/collegelogo"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:layout_centerHorizontal="true"
        android:layout_margin="5dp"
        android:background="#FFFFFFFF"
        android:contentDescription="college logo image"
        android:foregroundGravity="center_horizontal"
        android:padding="5dp"
        android:src="@drawable/collegelogo_transparent" />

    <TextView
        android:id="@+id/cs102Title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:padding="5dp"
        android:text="@string/cs102_title"
        android:textAlignment="center"
        android:textColor="@color/text_color"
        android:textSize="25sp"
        android:textStyle="bold" />
</RelativeLayout>
```

Gradle build finished in 3s 899ms (16 minutes ago)

+ Compile and run your app on ADV to see the result:



<u>EXPERIMENT:</u>
Change the "alignment" property of ImageView in activity_main.xml layout to move it the bottom of screen:



Step 2: Set "Click listener" for ImageView to interact with users:
When users touch the ImageView, ImageView will display grenoble.jpg image instead:

+ Open to MainActivity.java, keep the existing codes:

_Right below the class declaration, declare a variable "myImageView" from ImageView class:

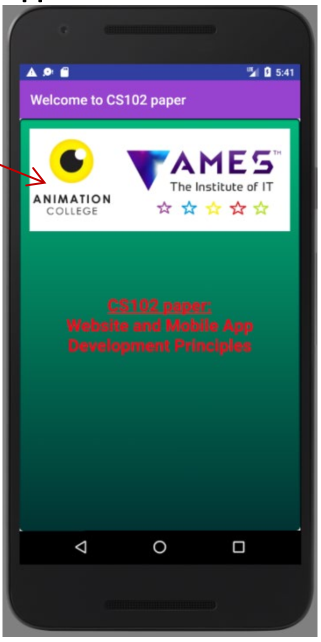
```
//Step 1: Declare a "myImageView" variable
private ImageView myImageView;
```

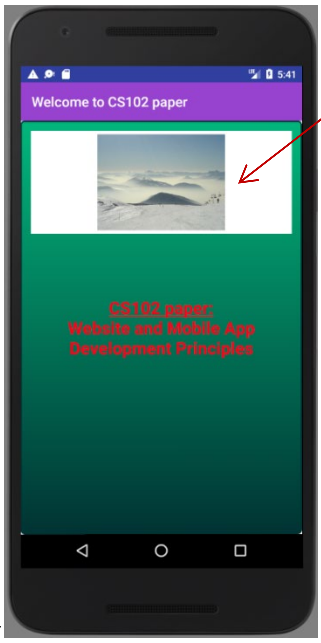
_Inside onCreate() method, add the below java code:

```
/////////////////////////////////////////////////////////////////////////////
//When users touch ImageView, ImageView will display grenoble.jpg image
//Step 2: Find the reference of the "myImageView" variable to UI element ("collegelogo")
//to make the connection between UI element ("collegelog") and variable ("myImageView") in java code
myImageView = (ImageView) findViewById(R.id.collegelogo);
//Step 3: Set "Click Listener" for "myImageView" variable
myImageView.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Step 4: When users click ImageView has been detected
        //Change the "src" property to grenoble.jpg image
        myImageView.setImageResource(R.drawable.grenoble);
    }
});
```

+ Compile and run your app on AVD to see the result:

Touch ImageView





When users click ImageView, it will display grenoble.jpg image instead of college logo

+ You can see that the background of the ImageView is "WHITE" solid color. When users click on ImageView, we will change the background color to Transparent and set new source of image is "grenoble.png".

_Create a new color variable called "transparent" (color code = #00000000) inside colors.xml:

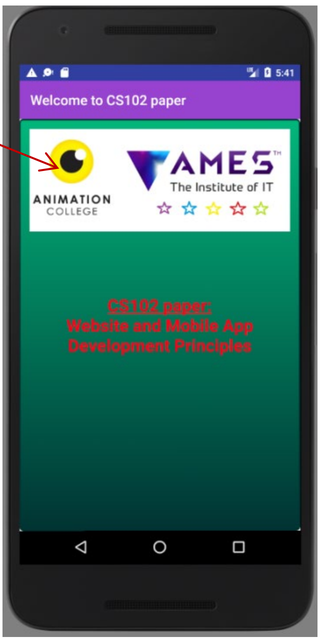
```
<!--Add "transparent" variable to define transparent color-->
<color name="transparent">#00000000</color>
```

_Inside onCreate() method, go to the section "set Click Listener for myImageView" variable, add a code line to set background color for myImageView:

```
//Change the background color to transparent
myImageView.setBackgroundResource(R.color.transparent);
```

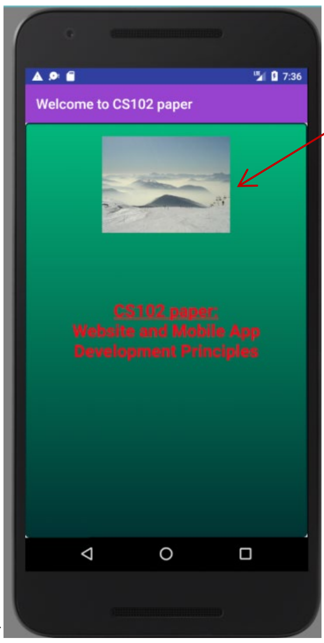
+ Compile and run your app on AVD to see the result:

Touch ImageView


Welcome to CS102 paper

CS102 paper:
Website and Mobile App Development Principles

→


Welcome to CS102 paper

CS102 paper:
Website and Mobile App Development Principles

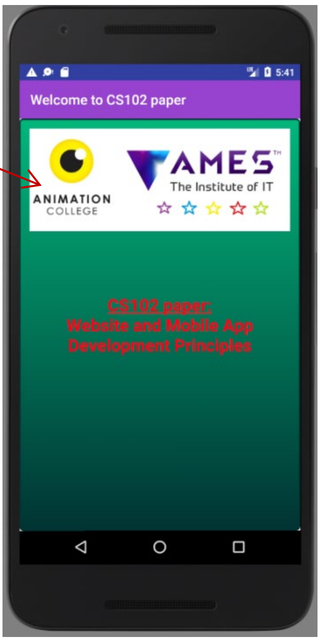
When users click ImageView, it will display grenoble.jpg image with transparent background

EXPERIMENT:

Modify the code a little bit so that when users click ImageView (college logo), the app will:

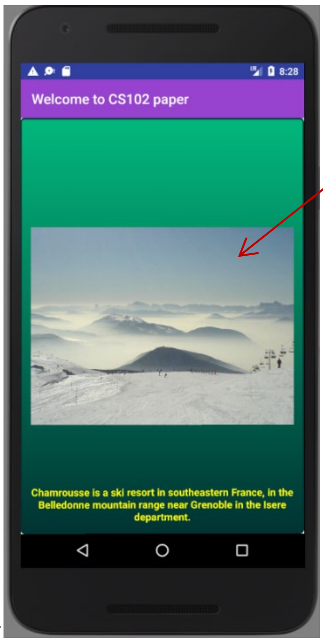
- Change the college logo image to “grenoble” image & set ImageView background color is transparent;
- Move the ImageView to center of screen;
- Change the TextView content to “Chamrousse is a ski resort in southeastern France, in the Belledonne mountain range near Grenoble in the Isere department.”
- Set TextView color to YELLOW & Set TextSize to 15.0f;
- Move the TextView to be located at the bottom of screen;

Touch ImageView


Welcome to CS102 paper

CS102 paper:
Website and Mobile App Development Principles

→


Welcome to CS102 paper

Chamrousse is a ski resort in southeastern France, in the Belledonne mountain range near Grenoble in the Isere department.

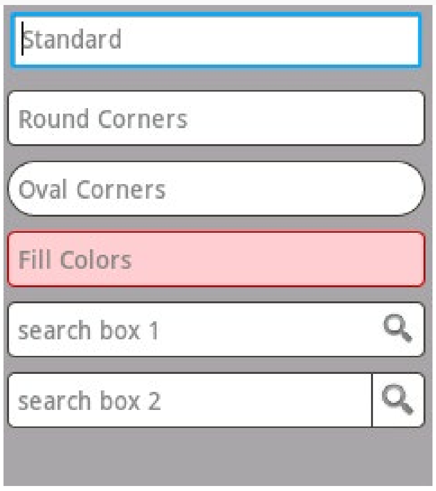
When users click ImageView, it will display grenoble.jpg image with transparent background at the centre of screen. The TextView changes text color, text size, text content and move to the bottom of screen.

PART 6: ADD EDITTEXT

A SUBCLASS OF "VIEW" SUPERCLASS

EditText:

Source: https://www.tutorialspoint.com/android/android_edittext_control.htm
An EditText is an overlay over TextView that configures itself to be **editable**. It is the predefined subclass of TextView that includes rich editing capabilities.



STYLES OF EDIT TEXT

EditText Attributes

Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.
Inherited from **android.widget.TextView** Class:

	Attribute & Description
1	**android:autoText:** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom:** This is the drawable to be drawn below the text.
3	**android:drawableRight:** This is the drawable to be drawn to the right of the text.
4	**android:editable:** If set, specifies that this TextView has an input method.
5	**android:text:** This is the Text to display.

Inherited from **android.view.View** Class:

	Attribute & Description
1	**android:background:** This is a drawable to use as the background.
2	**android:contentDescription:** This defines text that briefly describes content of the view.
3	**android:id:** This supplies an identifier name for this view.
4	**android:onClick:** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility:** This controls the initial visibility of the view.

Step 1: Add an EditText on layout for users to enter their name

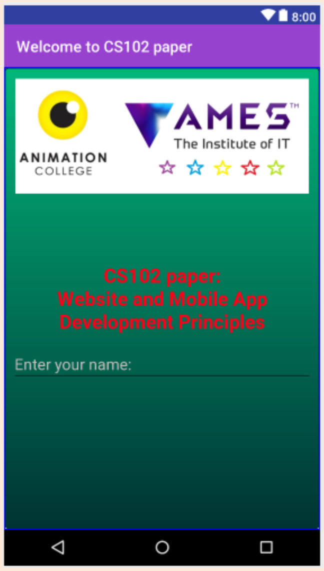
In this part, we will add **an EditText** for users to enter their name on screen.

+ Open activity_main.xml file and add an EditText to the layout, locating right below TextView element, set its id as "nameField", set hint "Enter your name: ", and set inputType as "text" as below:

```
<EditText
    android:id="@+id/nameField"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/cs102Title"
    android:layout_marginTop="10dp"
    android:hint="Enter your name:  "
    android:inputType="text"
    android:textColor="#FF0000FF"
    android:textColorHint="#cac1c1"
    android:textSize="20dp" />
```

+ The activity_main.xml layout file looks like:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/app_background"
    android:padding="10dp"
    tools:context=".MainActivity">

    <!--Add an ImageView to display the college logo image-->
    <ImageView
        android:id="@+id/collegelogo"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_margin="5dp"
        android:background="#FFFFFFFF"
        android:contentDescription="college logo image"
        android:foregroundGravity="center_horizontal"
        android:padding="5dp"
        android:scaleType="fitCenter"
        android:src="@drawable/collegelogo_transparent" />

    <!--Add a TextView to display cs102 title-->
    <TextView
        android:id="@+id/cs102Title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:padding="5dp"
        android:text="@string/cs102_title"
        android:textAlignment="center"
        android:textColor="@color/text_color"
        android:textSize="25sp"
        android:textStyle="bold" />

    <!--Add an EditText for users to enter their name-->
    <EditText
        android:id="@+id/nameField"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/cs102Title"
        android:layout_marginTop="10dp"
        android:hint="Enter your name:  "
        android:inputType="text"
        android:textColor="#FF0000FF"
        android:textColorHint="#cac1c1"
        android:textSize="20dp" />
</RelativeLayout>
```


8:00

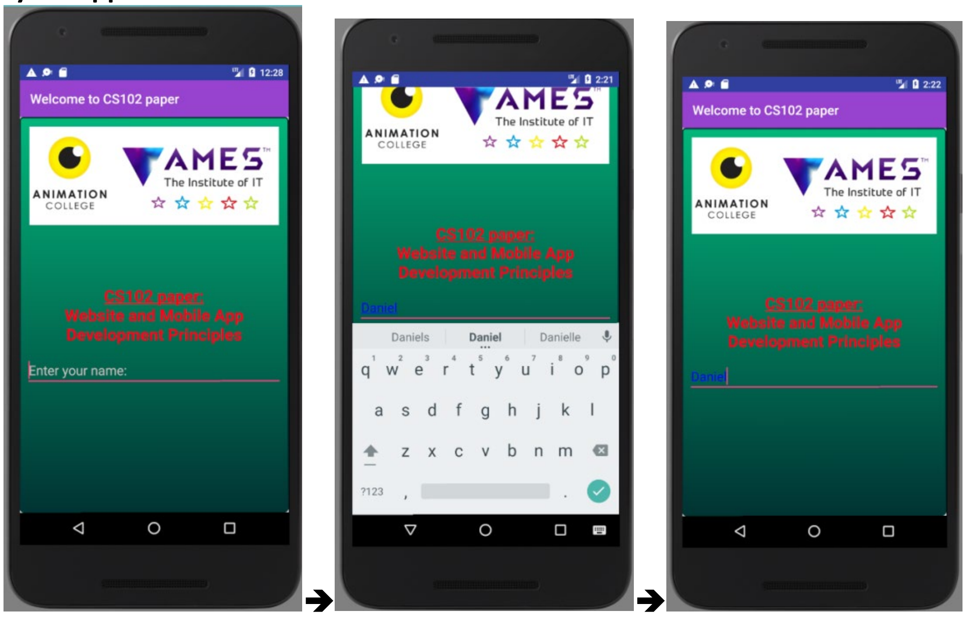
Welcome to CS102 paper

CS102 paper:
Website and Mobile App
Development Principles

Enter your name:

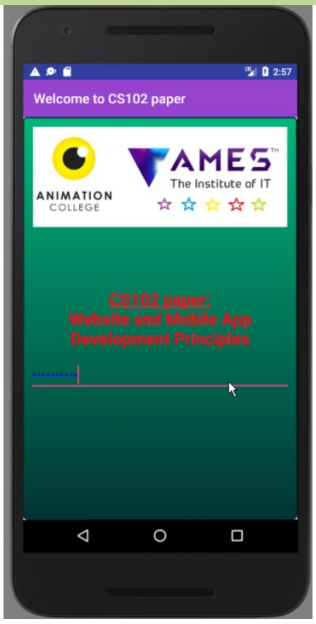
UI (layout) "Preview"

+ Compile and run your app on ADV to see the result:



EXPERIMENT:

Change the "inputType" property of EditText from "text" to "textPassword" and run the app to see:



PART 7: ADD BUTTON

A SUBCLASS OF "VIEW" SUPERCLASS

Button control

A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.

Button Attributes

Following are the important attributes related to Button control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

Sr.No	Attribute & Description
1	**android:autoText:** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom:** This is the drawable to be drawn below the text.
3	**android:drawableRight:** This is the drawable to be drawn to the right of the text.
4	**android:editable:** If set, specifies that this TextView has an input method.
5	**android:text:** This is the Text to display.

Inherited from **android.view.View** Class:

Attribute	Description
1	**android:background:** This is a drawable to use as the background.
2	**android:contentDescription:** This defines text that briefly describes content of the view.
3	**android:id:** This supplies an identifier name for this view.
4	**android:onClick:** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility:** This controls the initial visibility of the view.

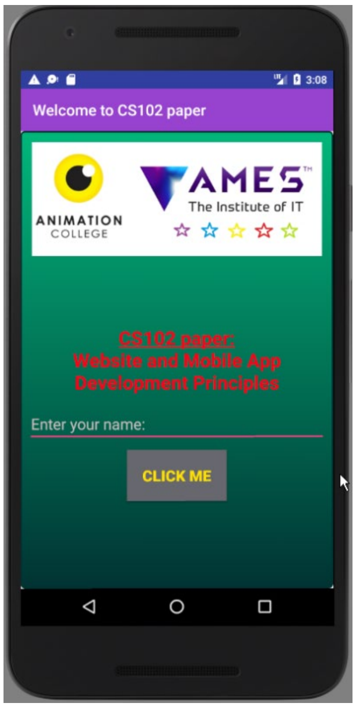
Step 1: Add a Button on layout

In this part, we will add **a Button** on screen.

+ Open activity_main.xml file and add a Buttont to the layout, locating right below EditText element, set its id as "clickMe", central to horizontal and set its text "Click me" as below:

```
<!--Add a Button "click me"-->
<Button
    android:id="@+id/clickMe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/nameField"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10dp"
    android:background="#67676e"
    android:padding="20dp"
    android:text="Click me"
    android:textColor="#ffe100"
    android:textSize="20sp"
    android:textStyle="bold" />
```

+ Compile and run your app on ADV to see the result:



EXPERIMENT:
Change the "alignment" property of Button in activity_main.xml layout to move it the bottom of screen.

Alert Dialog in Android

Source: https://www.tutorialspoint.com/android/android_alert_dialoges.htm
A Dialog is small window that prompts the user to a decision or enter additional information.

Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog.

Step 2: Set "Click listener" for Button to interact with users: When users click the Button, pop up an AlertDialog saying welcome message containing "Hello Daniel. Have a good day!" where "Daniel" is name entered inside EditText

+ Open to MainActivity.java, keep the existing codes:

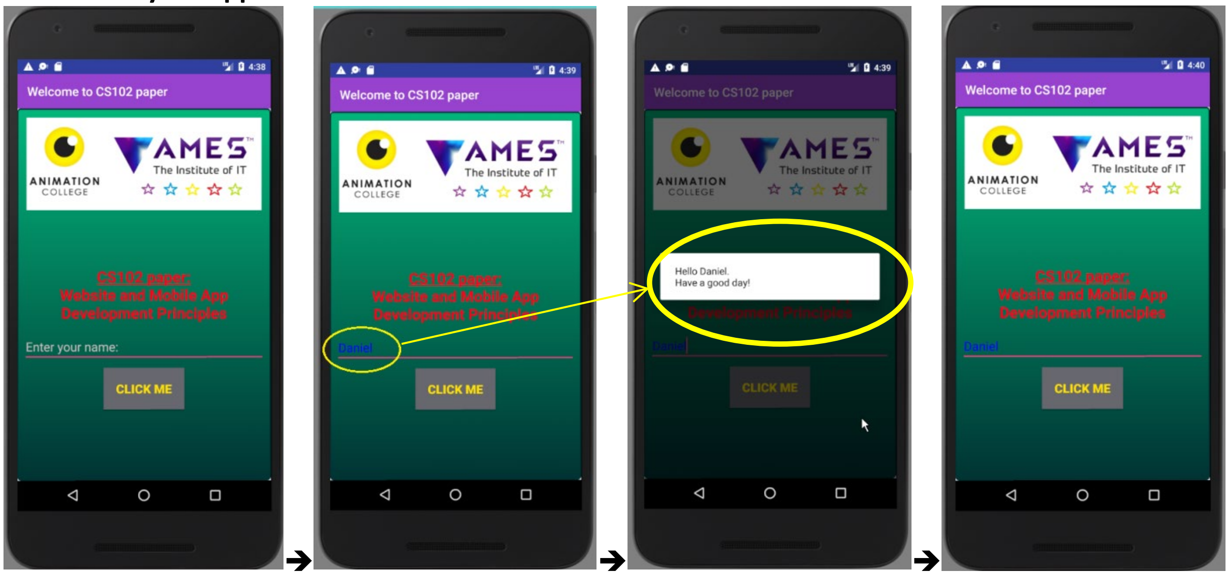
_Right below the class declaration, declare a variable "myButton" from Button class:

```java
//Step 1: Declare  two variables: "myButton" & "nameField"
private Button myButton;//Button
private EditText nameField;//EditText
```

_Inside onCreate() method, add the below java code:

```java
///////////////////////////////////////////////////////////////////////////////
//When users click the Button, pop up an AlertDialog saying welcome message containing
// "Hello" & entered name in the EditText
//Step 2: Find the reference of the "myButton" & "nameField" variables to UI elements
myButton = (Button) findViewById(R.id.clickMe);
nameField = (EditText) findViewById(R.id.nameField);
//Step 3: Set "Click Listener" for "myButton" variable
myButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Step 4: When users click Button, pop up alertDialog to show the welcome message
        //1: Get the entered name inside EditText by calling getText() method and toString() method
        String enteredName = nameField.getText().toString();
        //2: Build "welcomeMessage" string containing enteredName
        String welcomeMessage = "Hello " + enteredName + ".\nHave a good day!";
        //3: Create a "dialogBuilder" by using AlertDialog.Builder() method
        AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(MainActivity.this);
        //4: Assign the "welcomeMessage" variable to "dialogBuilder"
        dialogBuilder.setMessage(welcomeMessage);
        //5: Create a "dialog" object from "DialogBuilder"
        AlertDialog dialog = dialogBuilder.create();
        //6: Display "dialog" on screen
        dialog.show();
    }
});
```

+ Compile and run your app on AVD to see the result:



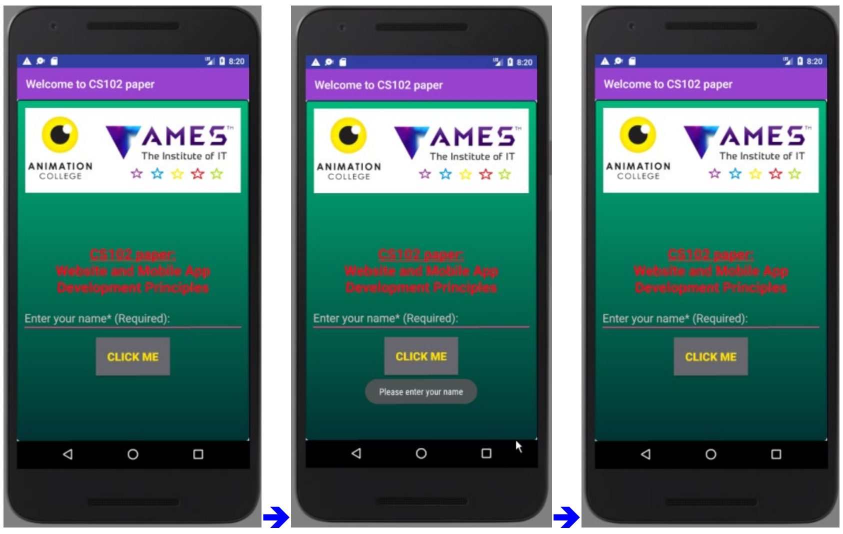
Hello Daniel.
Have a good day!

EXPERIMENT:
Add java code in MainActivity.java so that the app will check whether users have been entered their name in the EditText:

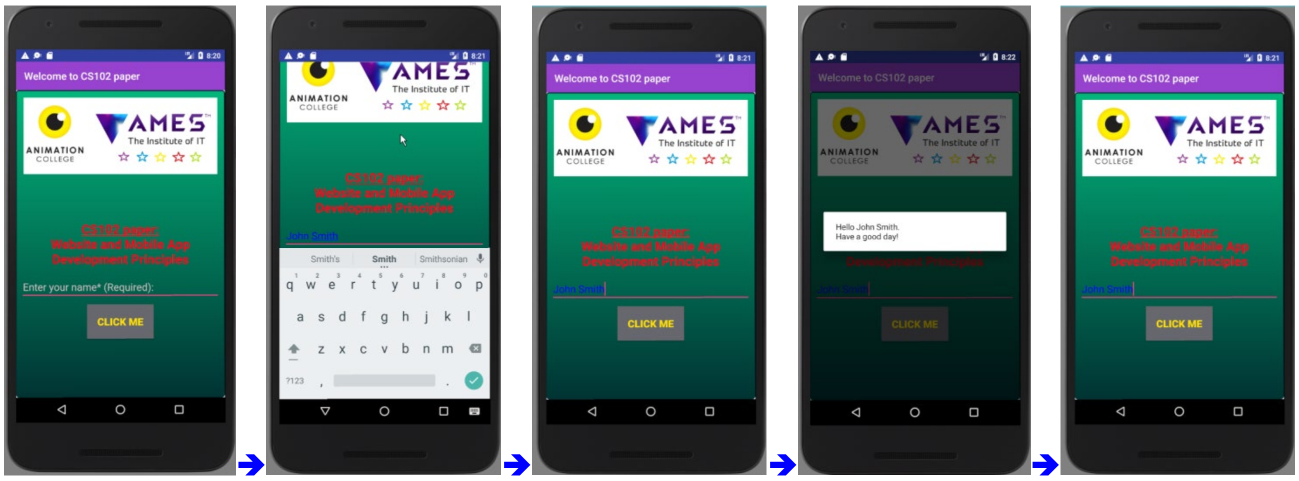
- If users click Button but haven't inputted their name, pop up a Toast message saying "Please enter your name" and don't display a Dialog with welcome message;

- If users click Button and also having entered their name, pop up a Dialog with welcome message containing **“Hello Daniel. Have a good day!”** where “Daniel” is name entered inside EditText.
- Change the “hint” property” of EditText to **“Enter your name * (required)”**

Hint: to check a string is empty or not, use function: **enteredName.isEmpty()**



If users click Button but haven’t inputted their name, pop up a Toast message saying “**Please enter your name**” and **don’t display a Dialog with welcome message**;



Welcome to CS102 paper
ANIMATION COLLEGE
AMES The Institute of IT
CS102 paper: Website and Mobile App Development Principles
Enter your name* (Required):
John Smith
CLICK ME
Hello John Smith. Have a good day!

If users click Button and also having entered their name, pop up a Dialog with welcome message containing **“Hello John Smith. Have a good day!”** where “**John Smith**” is name entered inside EditText.

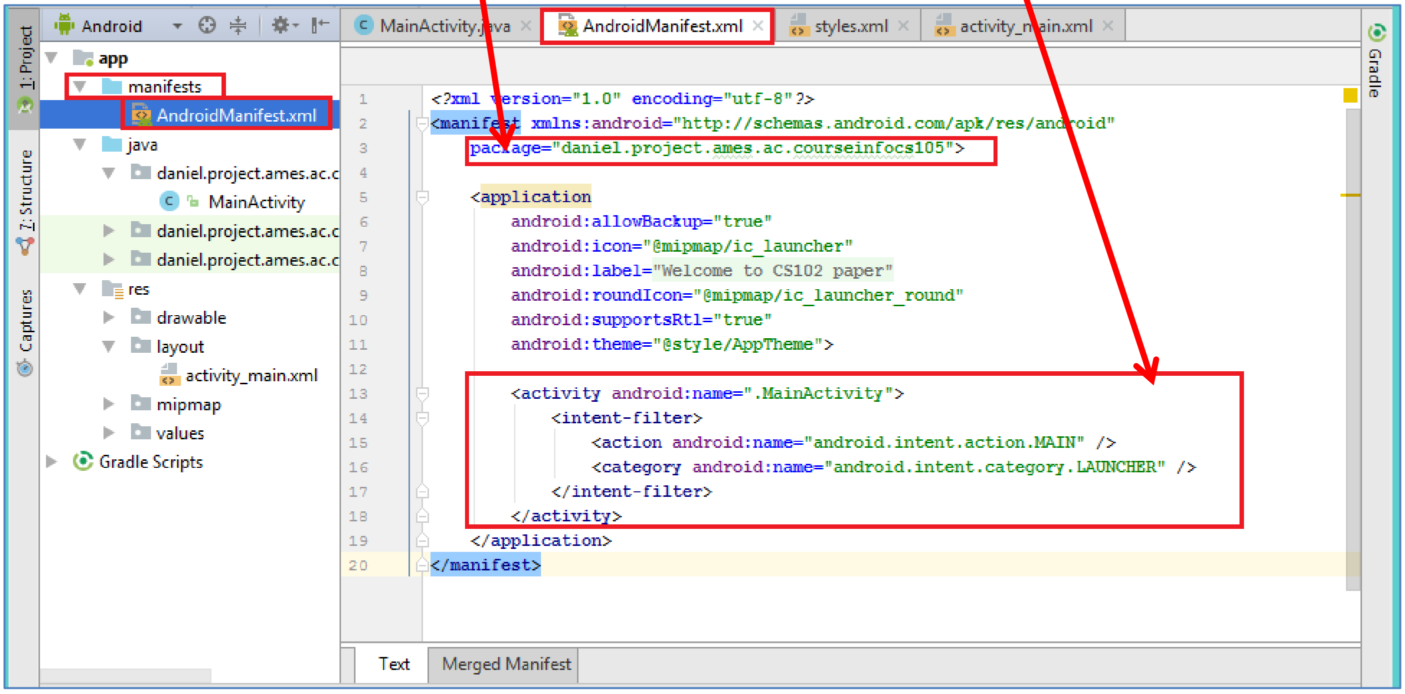
PART 8: Investigate Android Manifest

Open AndroidManifest.xml and have a look, don't change anything:

+ **AndroidManifest.xml** is a powerful file in the Android platform that allows you to describe the functionality and requirements of your application to Android.

+ **Every application must have an AndroidManifest.xml file** (with precisely that name) in its root directory. The **manifest** presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. Among other things, the manifest 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 — the activities, services, broadcast receivers, and content providers that the application is composed of.** It names the **classes that implement each of the components** and publishes their capabilities (for example, which **Intent messages** they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched.
- **It determines which processes** will host application components.
- **It declares which permissions 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 is running. These declarations are present in the manifest only while the application is being developed and tested; they're 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**.



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="daniel.project.ames.ac.courseinfocs105">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Welcome to CS102 paper"
        android:roundIcon="@mipmap/ic_launcher_round"
        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>
    </application>
</manifest>
```

End of Lab!

YOU DID A

GOOG JOB.

Congratulation!

Self-directed learning

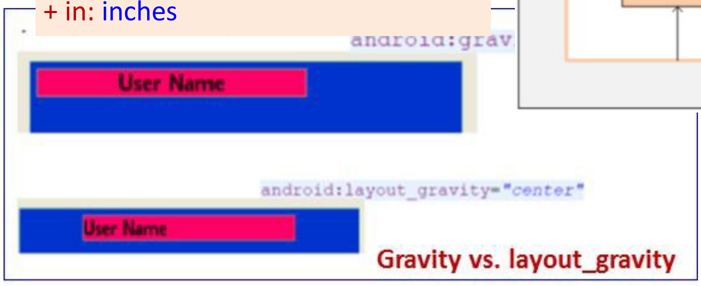
Investigate about Android UI design: UI Layout, UI controls, Themes, Style.

- Investigate other Android properties (XML): padding, margin, gravity, match_parent, wrap_content, ...
- Investigate other layouts: LinearLayout, TableLayout, AbsoluteLayout, FrameLayout, etc.

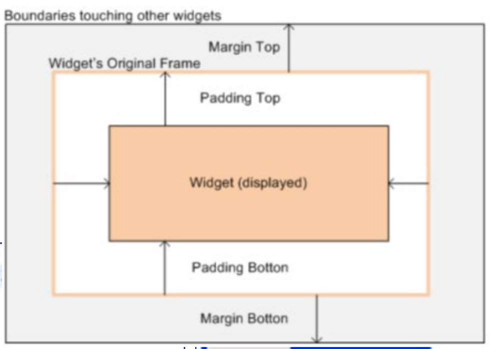
Self-directed learning

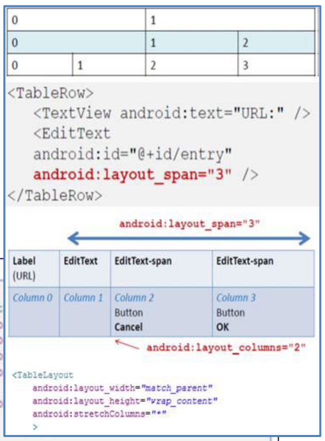
Android - Graphic User Interface (UI)

+ dp: Density-independent Pixels
+ sp: Scale-independent Pixels
+ pt: Points which is 1/72 of an inch
+ px: Pixels
+ mm: Millimeters
+ in: inches



Gravity vs. layout_gravity





0		1	
0		1	2
0	1	2	3

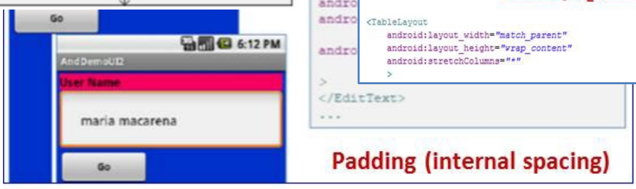
```
<TableRow>
   <TextView android:text="URL:" />
   <EditText
   android:id="@+id/entry"
   android:layout_span="3" />
</TableRow>
```

android:layout_span="3"

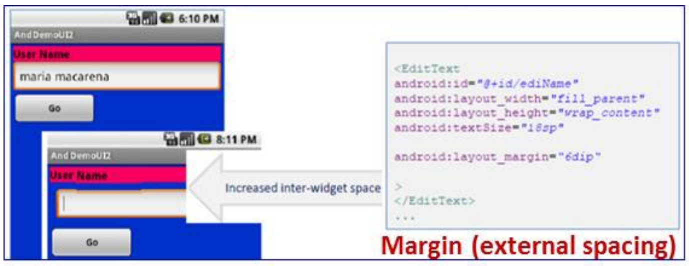
Label (URL)	EditText	EditText-span	EditText-span
Column 0	Column 1	Column 2 Button Cancel	Column 3 Button OK

android:layout_columns="2"

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="*"
    >
```

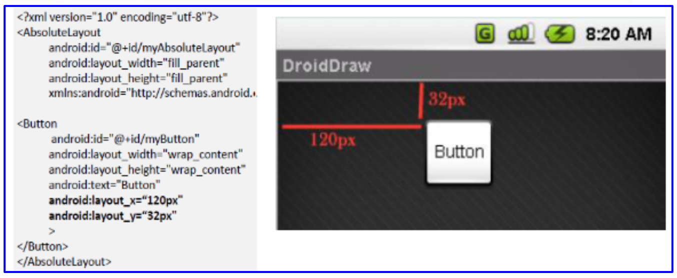


Padding (internal spacing)

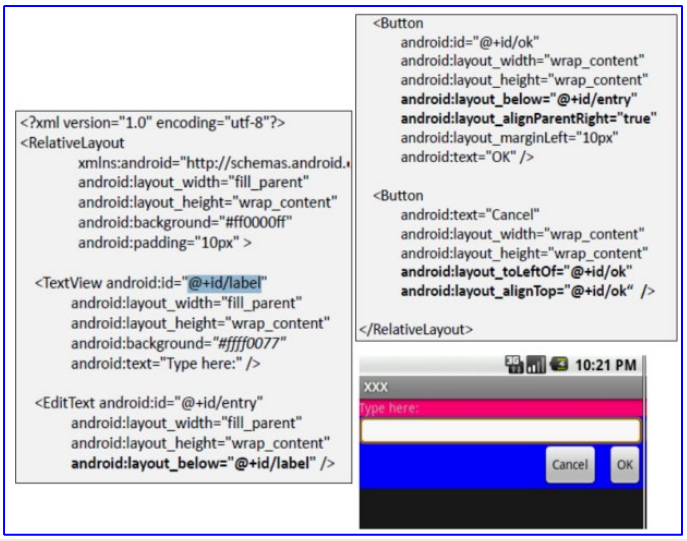


```
<EditText
android:id="@+id/ediName"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textSize="18sp"

android:layout_margin="6dip"

>
</EditText>
...
```

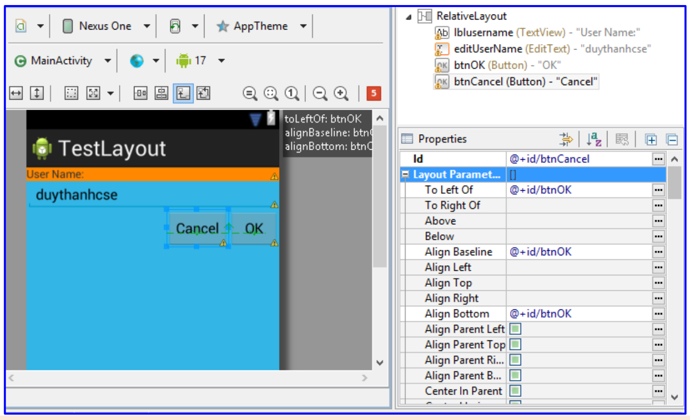
Margin (external spacing)



```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
        android:id="@+id/myAbsoluteLayout"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        xmlns:android="http://schemas.android.

<Button
         android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_x="120px"
        android:layout_y="32px"
        >
</Button>
</AbsoluteLayout>
```



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff0000ff"
        android:padding="10px" >

    <TextView android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0077"
        android:text="Type here:" />

    <EditText android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/label" />

  <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10px"
        android:text="OK" />

  <Button
        android:text="Cancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@+id/ok"
        android:layout_alignTop="@+id/ok“ />

</RelativeLayout>
```



```
RelativeLayout
    lblusername (TextView) - "User Name:"
    editUserName (EditText) - "duythanhcse"
    btnOK (Button) - "OK"
    btnCancel (Button) - "Cancel"
```

Properties

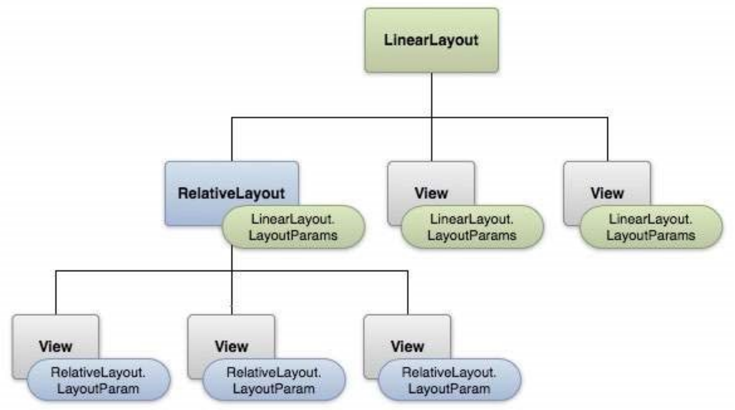
Id	@+id/btnCancel
Layout Paramet...	[]
To Left Of	@+id/btnOK
To Right Of	
Above	
Below	
Align Baseline	@+id/btnOK
Align Left	
Align Top	
Align Right	
Align Bottom	@+id/btnOK
Align Parent Left	☐
Align Parent Top	☐
Align Parent Ri...	☐
Align Parent B...	☐
Center In Parent	☐

Reading:
+ Android - UI design: https://www.tutorialspoint.com/android/android_ui_design.htm
+ Android - UI Layout: https://www.tutorialspoint.com/android/android_user_interface_layouts.htm
+ Android - UI controls: https://www.tutorialspoint.com/android/android_user_interface_controls.htm
+ Android - Themes and styles: https://www.tutorialspoint.com/android/android_styles_and_themes.htm

Android - UI Layouts

Source: https://www.tutorialspoint.com/android/android_user_interface_layouts.htm

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, text fields, etc.
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.



LAYOUT PARAMS

This tutorial is more about creating your GUI based on layouts defined in XML file. A layout may contain any type of widgets such as buttons, labels, textboxes, and so on. Following is a simple example of XML file having LinearLayout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
```

```
    <TextView android:id="@+id/text"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="This is a TextView" />

    <Button android:id="@+id/button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="This is a Button" />

    <!-- More GUI components go here  -->

</LinearLayout>
```

Once your layout has created, you can load the layout resource from your application code, in your *Activity.onCreate()* callback implementation as shown below:

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
}
```

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	**Linear Layout**: LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	**Relative Layout**: RelativeLayout is a view group that displays child views in relative positions.
3	**Table Layout**: TableLayout is a view that groups views into rows and columns.
4	**Absolute Layout**: AbsoluteLayout enables you to specify the exact location of its children.
5	**Frame Layout**: The FrameLayout is a placeholder on screen that you can use to display a single view.
6	**List View**: ListView is a view group that displays a list of scrollable items.
7	**Grid View**: GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the view.
2	**android:layout_width:** This is the width of the layout.
3	**android:layout_height:** This is the height of the layout
4	**android:layout_marginTop:** This is the extra space on the top side of the layout.
5	**android:layout_marginBottom:** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft:** This is the extra space on the left side of the layout.
7	**android:layout_marginRight:** This is the extra space on the right side of the layout.
8	**android:layout_gravity:** This specifies how child Views are positioned.
9	**android:layout_weight:** This specifies how much of the extra space in the layout should be allocated to the

	View.
10	**android:layout_x:** This specifies the x-coordinate of the layout.
11	**android:layout_y:** This specifies the y-coordinate of the layout.
12	**android:layout_width:** This is the width of the layout.
13	**android:layout_width:** This is the width of the layout.
14	**android:paddingLeft:** This is the left padding filled for the layout.
15	**android:paddingRight:** This is the right padding filled for the layout.
16	**android:paddingTop:** This is the top padding filled for the layout.
17	**android:paddingBottom:** This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px (Pixels), mm (Millimeters) and finally in (inches).
You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height:

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

Constant	Value	Description
top	0x30	Push object to the top of its container, not changing its size.
bottom	0x50	Push object to the bottom of its container, not changing its size.
left	0x03	Push object to the left of its container, not changing its size.
right	0x05	Push object to the right of its container, not changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.
fill	0x77	Grow the horizontal and vertical size of the object if needed so it completely fills its container.
clip_vertical	0x80	Additional option that can be set to have the top and/or bottom edges of the child clipped to its container's bounds. The clip will be based on the vertical gravity: a top gravity will clip the bottom edge, a bottom gravity will clip the top edge, and neither will clip both edges.
clip_horizontal	0x08	Additional option that can be set to have the left and/or right edges of the child clipped to its container's bounds. The clip will be based on the horizontal gravity: a left gravity will clip the right edge, a right gravity will clip the left edge, and neither will clip both edges.
start	0x00800003	Push object to the beginning of its container, not changing its size.
end	0x00800005	Push object to the end of its container, not changing its size.

View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

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

Following is a brief description of @ and + signs –

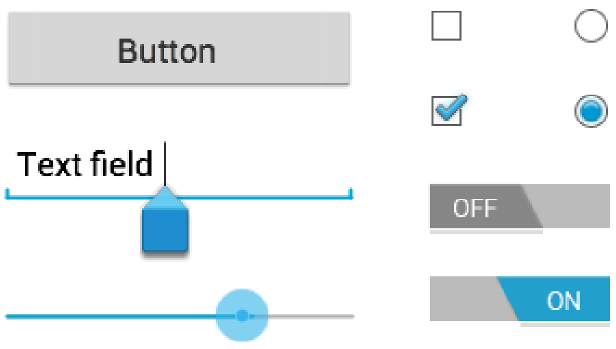
- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following:

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Android UI Controls

Source: UI controls: https://www.tutorialspoint.com/android/android_user_interface_controls.htm

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.



UI ELEMENTS

A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

You define your layout in an XML file which offers a human-readable structure for the layout, similar to HTML. For example, a simple vertical layout with a text view and a button looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />

  <Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a Button" />

</LinearLayout>
```

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description
1	**TextView**: This control is used to display text to the user.
2	**EditText**: EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	**AutoCompleteTextView**: The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	**Button**: A push-button that can be pressed, or clicked, by the user to perform an action.
5	**ImageButton**: An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	**CheckBox**: An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	**ToggleButton**: An on/off button with a light indicator.
8	**RadioButton**: The RadioButton has two states: either checked or unchecked.
9	**RadioGroup**: A RadioGroup is used to group together one or more RadioButtons.
10	**ProgressBar**: The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	**Spinner**: A drop-down list that allows users to select one value from a set.
12	**TimePicker**: The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	**DatePicker**: The DatePicker view enables users to select a date of the day.

Create UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is:

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

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView android:id="@+id/text_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />
</LinearLayout>
```

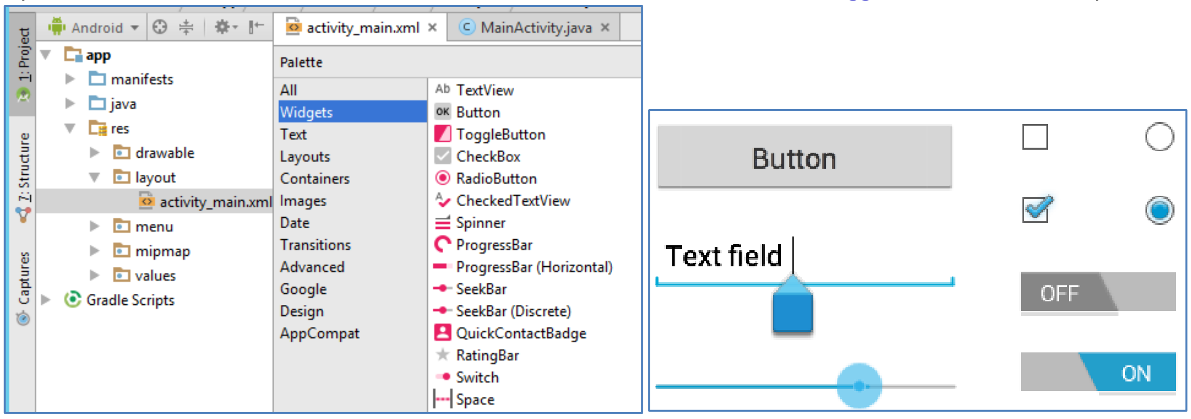
Then finally create an instance of the Control object and capture it from the layout, use the following:

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

Other Graphic UI elements/widgets in Android – “View” superclass

Source: https://www.tutorialspoint.com/android/android_user_interface_controls.htm

The “View” class represents the **basic building block for user interface components**. A View 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 (**Button, TextView, ImageView, TextClock**, etc.).
Input controls are the **interactive components in your app's user interface**. Android provides a wide variety of controls you can use in your UI, such as **buttons**, **text fields**, **seek bars**, **check box**, **zoom buttons**, **toggle buttons**, and many more.



UI ELEMENTS

Once you have created **Views**, there are typically a few types of common operations you may wish to perform:

- **Set properties**: for example setting the text of a TextView. The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.
- **Set focus:** The framework will handle moving focus in response to user input. To force focus to a specific view, call requestFocus().
- **Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using setOnFocusChangeListener(android.view.View.OnFocusChangeListener). Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.
- **Set visibility:** You can hide or show views using setVisibility(int).

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

	UI Control & Description
1	**TextView**: This control is used to display text to the user.
2	**EditText**: EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	**AutoCompleteTextView**: The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	**Button**: A push-button that can be pressed, or clicked, by the user to perform an action.
5	**ImageButton**: An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	**CheckBox**: An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	**ToggleButton**: An on/off button with a light indicator.
8	**RadioButton**: The RadioButton has two states: either checked or unchecked.
9	**RadioGroup**: A RadioGroup is used to group together one or more RadioButtons.
10	**ProgressBar**: The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	**Spinner**: A drop-down list that allows users to select one value from a set.
12	**TimePicker**: The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	**DatePicker**: The DatePicker view enables users to select a date of the day.

Android - Styles and Themes

Source: https://www.tutorialspoint.com/android/android_styles_and_themes.htm

A **style** resource defines the format and look for a UI. A style can be applied to an individual View (from within a layout file) or to an entire Activity or application (from within the manifest file).

Defining Styles

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under **res/values/** directory of your project and will have **<resources>** as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.

You can define multiple styles per file using **<style>** tag but each style will have its name that uniquely identifies the style. Android style attributes are set using **<item>** tag as shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomFontStyle">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:capitalize">characters</item>
    <item name="android:typeface">monospace</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textColor">#00FF00</item>/>
  </style>
</resources>
```

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.

Using Styles

Once your style is defined, you can use it in your XML Layout file using **style** attribute as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/text_id"
    style="@style/CustomFontStyle"
    android:text="@string/hello_world" />

</LinearLayout>
```

Style Inheritance

Android supports style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add.
To implement a custom theme create or edit **/res/values/themes.xml** and add the following:

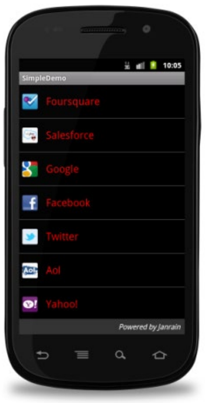
```
<resources>
  ...
  <style name="MyCustomTheme" parent="android:style/Theme">
  <item name="android:textColorPrimary">#ffff0000</item>
  </style>
  ...
</resources>
```

In your **AndroidManifest.xml** apply the theme to the activities you want to style –

```
<activity
```

```
android:name="com.myapp.MyActivity"
...
android:theme="@style/MyCustomTheme"
/>
```

Your **new theme** will be applied to your activity, and text is now bright red.

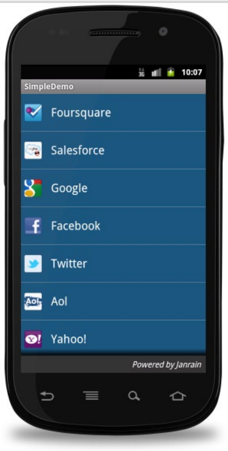


Applying Colors to Theme Attributes

Your color resource can then be applied to some theme attributes, such as the window background and the primary text color, by adding <item> elements to your custom theme. These attributes are defined in your styles.xml file.

For example, to apply the custom color to the window background, add the following two <item> elements to your custom theme, defined in **MyAndroidApp/res/values/styles.xml** file:

```
<resources>
  ...
  <style name="MyCustomTheme" ...>
    <item name="android:windowBackground">@color/my_custom_color</item>
    <item name="android:colorBackgroundCacheHint">@color/my_custom_color</item>
  </style>
  ...
</resources>
```



Android Themes

Hope you understood the concept of Style, so now let's try to understand what is a **Theme**. A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.

Thus, 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 **CustomFontStyle** 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">
```

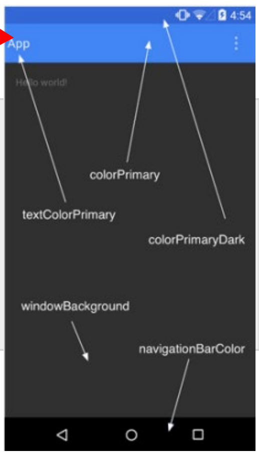
There are number of default themes defined by Android which you can use directly or inherit them using **parent** attribute as follows –

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

To understand the concept related to Android Theme, you can check Theme Demo Example.

Styling the colour palette

The layout design can implementable based on them based colours, for example as beside design is designed based on them **colour (blue)**.



The 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>
```

Default Styles & Themes

The Android platform provides a large collection of styles and themes that you can use in your applications. You can find a reference of all available styles in the **R.style** class.

To use the styles listed here, replace all underscores in the style name with a period. For example, you can apply the **Theme_NoTitleBar** theme with **"@android:style/Theme.NoTitleBar**".

You can see the following source code for Android styles and themes:

- Android Styles (styles.xml)
- Android Themes (themes.xml)