**Your First Android App in Visual Studio**

The **Android** operating system is based on the Dalvik Virtual Machine (VM), which is a mobile-optimised VM similar to the Java VM. Oxygene for Java ships with templates for creating Android projects, and produces both native Java JAR files and the Android APK file necessary for deployment to an Android device.

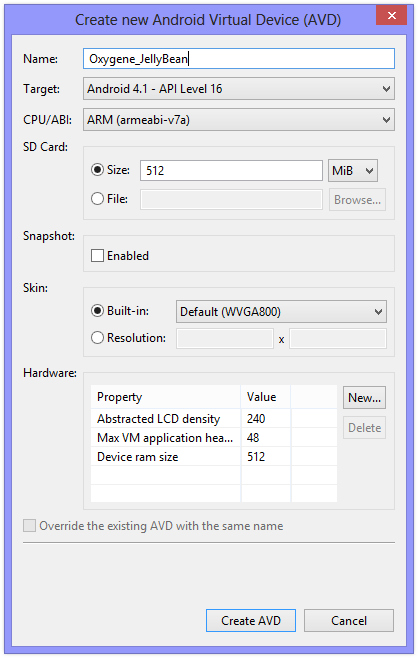
Because Android runs on such a wide variety of devices with different screen sizes, formats and orientations, it was intentionally designed without a WYSIWYG design surface for building the UI. Instead, an XML file (similar to .NET’s XAML) is edited to lay out the visual elements. There is a free online [DroidDraw](http://www.droiddraw.org/) tool for building Android User Interfaces via a WYSIWYG interface, but editing the XML directly is recommended.

**Prerequisites and Emulators**

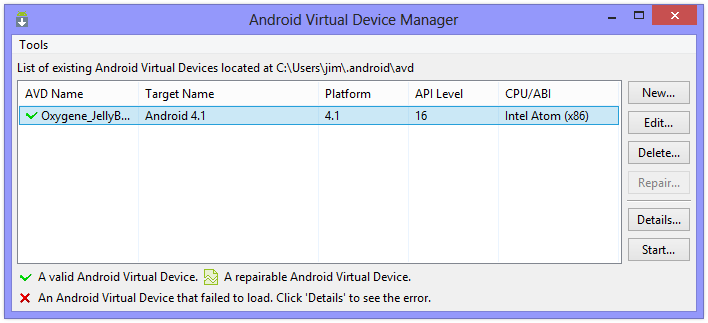
To get started with Android development, you need to install the Java Development Kit and Android SDK, as outlined [here](https://docs.elementscompiler.com/Fire/Setup/Mac/AndroidSDK/) (Fire) and here (Visual Studio).

When the tools and platforms are all installed, you will be able to create an [Android Emulator](http://developer.android.com/tools/devices/emulator.html), also known as an Android Virtual Device or AVD. You can do this from the Android Virtual Device Manager, which is accessible from the SDK Manager by choosing Tools, Manage AVDs.

Just click New, give the emulator a name and select the API in the Target field. You can choose any installed API level, for example Android 2.2 - API Level 8 (also known as Froyo) or Android 4.0.3 - API Level 15 (also known as Ice Cream Sandwich). Once you’ve specified the SD Card Size for the emulator (say 512) and chosen a skin (a screen resolution) you can use the Create AVD button to finish the job.

[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-Create_new_Android_Virtual_Device.png)

You can launch the emulator from this screen by selecting it and pressing the Start button.

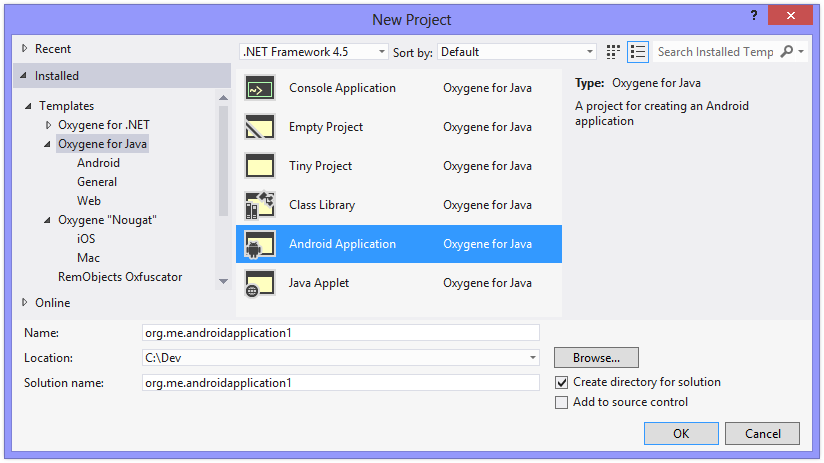
[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-Android_Virtual_Device_Manager.png)

**Note**: When you re-run the SDK Manager, it will often find updates to install. As mentioned earlier, if the Android SDK was installed into the default location, it will require administrative privileges to install them. So be sure to run it as Administrator (or install the Android SDK into a custom location to make things simpler).

The first time you create or open an Elements project for Android, it will do a ‘pre-flight check’ to ensure that it can locate the things it needs, notably the JDK and the Android SDK. If you’ve installed them into custom locations and it fails to find them, this gives you an opportunity to specify the installation folders.

**Getting Started**

In both Visual Studio and Fire, the New Project dialog provides the Android app template under Oxygene for Java and Android.

[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-Oxygene_for_Java_-_Android_-_New_Project.png)

The new Android project is created with a single simple activity called MainActivity. An *Activity* is the most basic part of an Android app – a single, focused thing that the user can do. The pre-created MainActivity contains a small amount of code to set up a simple UI with a button that, when clicked, displays an incrementing count on its caption.

The visual representation of the screen for MainActivity is defined in the XML file ''main.layout-xml'' which is in the ''res/layout'' folder.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical"

android:layout\_width="fill\_parent" android:layout\_height="fill\_parent"

android:gravity="center\_vertical">

<LinearLayout android:orientation="horizontal"

android:layout\_width="fill\_parent" android:layout\_height="wrap\_content"

android:gravity="center\_horizontal">

<Button android:id="@+id/MyButton" android:text="@string/my\_button\_text"

android:layout\_width="wrap\_content" android:layout\_height="wrap\_content"></Button>

</LinearLayout>

</LinearLayout>

Controls are named with the android:id attribute in layout XML. You prefix @+id/ in front of the chosen name and you can then reference the control’s ID in code by prefixing it with R.id. (all IDs become members of the id class, which is nested within the R resource class). To access the control, you can pass the ID into the activity’s [findViewById()](http://developer.android.com/reference/android/app/Activity.html#findViewById(int)) method. For example, the button named in the layout XML above has an ID accessible in code as R.id.MyButton. To get access to the button object you’d use Button(findViewById(R.id.MyButton)) - notice the typecast to get a [Button](http://developer.android.com/reference/android/widget/Button.html) object, which is needed because findViewById() returns a [View](http://developer.android.com/reference/android/view/View.html) object, one of the Button class’s ancestor types.

Strings referenced in the XML attributes with the @string/ prefix or in the code as members of the R.string class are defined in the ''strings.android-xml'' resource file, which can be found in the ''res/values'' folder.

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">org.me.androidapplication1</string>

<string name="my\_button\_text">Click Me!</string>

<string name="my\_button\_text\_2">%1$d clicks!</string>

</resources>

To reference the string resource in code from an activity method, you simply call getString(R.string.my\_button\_text). [getString()](http://developer.android.com/reference/android/content/Context.html#getString(int)) is a method of the [Activity](http://developer.android.com/reference/android/app/Activity.html) class (or, more accurately, a method of one of [Activity](http://developer.android.com/reference/android/app/Activity.html)’s ancestor classes, [Context](http://developer.android.com/reference/android/content/Context.html)). As you can see, you pass a string resource ID to the method and it returns the resultant corresponding string.

'''Note''': In Delphi and in .NET languages we are used to working with properties. A property represents some data with possible side effects when read from and/or when written to. A property Foo is defined in terms of a getFoo() getter function that returns a value and a setFoo() setter function that is passed a value. Java does not support the notion of properties, so classes have many getter and setter methods exposed instead of their equivalent properties. When working with Oxygene for Java, you have the choice of calling the getter/setter type methods that exist in any Java libraries that you reference, such as the Android SDK, or using the property that you might normally expect to exist. So in the case above, you can access a string resource either by calling:

type

MainActivity = public class(Activity)

private

Count: Integer := 0;

public

method onCreate(savedInstanceState: Bundle); override;

method ButtonOnClick(v: View);

end;

...

method MainActivity.onCreate(savedInstanceState: Bundle);

begin

inherited;

// Set our view from the "main" layout resource

ContentView := R.layout.main;

// Get our button from the layout resource,

// and attach an event to it

var myButton: Button := Button(findViewById(R.id.MyButton));

myButton.OnClickListener := new interface View.OnClickListener(onClick := @ButtonOnClick);

end;

method MainActivity.ButtonOnClick(v: View);

begin

inc(Count);

(v as Button).Text := WideString.format(String[R.string.my\_button\_text\_2], Count);

end;

[onCreate](http://developer.android.com/reference/android/app/Activity.html#onCreate) is the method called when the activity is first created and where your activity initialisation goes. After calling through to the inherited method, you can see that a layout resource ID is assigned to the ContentView property, although given the note above, it should be clear that really we are are passing the resource ID to [setContentView()](http://developer.android.com/reference/android/app/Activity.html#setContentView). This sets up the layout file ''res.layout-xml'' as the UI for this main activity.

Next the code locates the button with the ID MyButton and stores a reference to it in a local variable.

The final job of onCreate() is to set up the button’s click event handler, which is done by assigning an expression to the button’s OnClickListener property, or in truth passing it to the [setOnClickListener()](http://developer.android.com/reference/android/view/View.html#setOnClickListener) method. Because Java uses interfaces to define event signatures, we use Oxygene’s inline interface implementation to associate our ButtonOnClick method with the [onClick](http://developer.android.com/reference/android/view/View.OnClickListener.html#onClick) method of the button’s [View.OnClickListener](http://developer.android.com/reference/android/view/View.OnClickListener.html) event interface.

The event handler method itself, ButtonOnClick, increments the Count class instance variable and then uses the value to create a formatted string, which is then set as the button’s caption via its Text property (or [setText()](http://developer.android.com/reference/android/widget/TextView.html#setText) method). The string formatting uses the value of the my\_button\_text\_2 string resource (shown earlier), which uses Android [format string syntax](http://developer.android.com/reference/java/util/Formatter.html). The formatting method being called is really [String.format()](http://developer.android.com/reference/java/lang/String.html#format). It’s being called as WideString.format() to avoid ambiguity with the String property ([getString()](http://developer.android.com/reference/android/content/Context.html" \l "getString) method) of the Activity class we looked at just above. WideString is provided by Oxygene as a synonym for the [String](http://developer.android.com/reference/java/lang/String.html) type.

One really important value in the ''strings.android-xml'' file is the app\_name string. This is used twice by the application:

* the activity’s title bar has this string written on it and
* the list of installed apps on the device uses this string to identify the app.

Be sure to update app\_name to make it meaningful.

You can find the references to app\_name that affect the title bar and the installed app list in the [http://developer.android.com/guide/topics/manifest/manifest-intro.html Android manifest file], ''AndroidManifest.android-xml'' in the ''Properties'' folder. Every Android application has a manifest file to let Android know the identity of the application package, the components in the application, any permissions required in order to operate and some other system details.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="org.me.androidapplication1">

<application android:persistent="true"

android:label="@string/app\_name"

android:icon="@drawable/icon"

android:debuggable="true">

<activity android:label="@string/app\_name" android:name="org.me.androidapplication1.MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

<uses-sdk android:minSdkVersion="4" />

</manifest>

In this sample application’s manifest you can see the application is given a text label and an icon, and the single activity is identified by its class name and also given a label. The convoluted looking intent filter declaration inside this activity is simply the standard code necessary to tell Android this activity is the main activity of the app.

When you need to add more strings to your project, simply edit the ''strings.android-xml'' file.

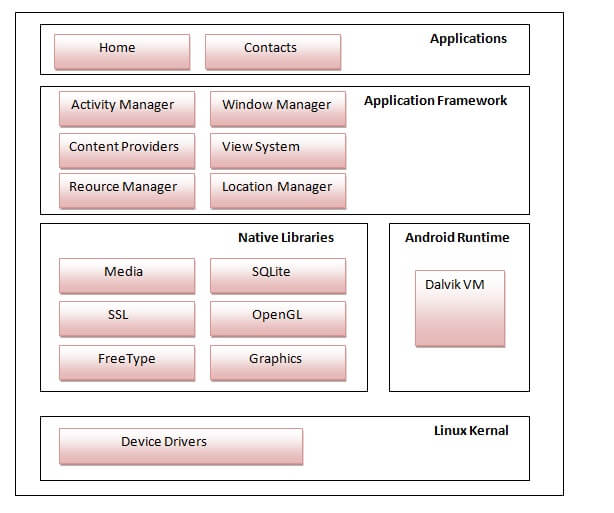
'''Note''': The various Android resource files that reside within the ''res'' folder hierarchy are all XML files. They would all be perfectly valid and usable if given the standard .xml extension. The reason for the .layout-xml and .android-xml extensions is to enable the appropriate IntelliSense (or Code Completion) behaviour when working in these files.

**Android Architecture**

**android architecture** or **Android software stack** is categorized into five parts:

1. linux kernel
2. native libraries (middleware),
3. Android Runtime
4. Application Framework
5. Applications

Let's see the android architecture first.



**1) Linux kernel**

It is the heart of android architecture that exists at the root of android architecture. **Linux kernel** is responsible for device drivers, power management, memory management, device management and resource access.

**2) Native Libraries**

On the top of linux kernel, their are **Native libraries** such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc.

The WebKit library is responsible for browser support, SQLite is for database, FreeType for font support, Media for playing and recording audio and video formats.

**3) Android Runtime**

In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

**4) Android Framework**

On the top of Native libraries and android runtime, there is android framework. Android framework includes **Android API's** such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

**5) Applications**

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernal.

**Running Your App**

When you build your app, the Oxygene compiler first creates a normal Java JAR file and then the Android toolchain creates an Android APK file (Android PacKage). The JAR file is just compiled code (Java byte code) but the APK file is the Java byte code processed to run on Android’s Dalvik Virtual Machine with all the necessary resources packaged into the file. Once the package is built, it is ready to run either on an AVD (in other words on the emulator) or on a physical device.

'''Note''': To deploy an APK file from your Windows machine to a physical device, such as a phone or tablet, you must first install the manufacturer’s USB driver for the device to allow communication between the PC and the device. Then you connect the device to the PC using an appropriate USB cable. You should be able to locate instructions on how to install the driver on your manufacturer’s web site or by using a search engine.

The Oxygene debugger will automatically deploy the APK to the emulator and launch it when you use the Start Debugging or Start Without Debugging buttons in Visual Studio (or use the F5 or Ctrl+F5 keystrokes, respectively). If you choose Start Debugging (or F5), you can set breakpoints and debug your application from Visual Studio as it runs on the emulator or physical device.

If you have a virtual Android device running and also have a physical Android device connected to the computer, you need some way to tell Oxygene which device to target when you start an application. You can choose an Android device in the Android section of the project properties.

If you want to install the APK manually, you can use the [Android Debug Bridge (ADB)](http://developer.android.com/guide/developing/tools/adb.html) command-line tool from a Windows command prompt. The adb.exe tool is located in the ''<Android\_SDK\_installation\_path>\android-sdk-windows\platform-tools'' folder, where ''<Android\_SDK\_installation\_path>'' could be ''C:\Program Files'' or another directory you chose at installation time. It may be of benefit to add both ''<Android\_SDK\_installation\_path>\android-sdk-windows\platform-tools'' and also ''<Android\_SDK\_installation\_path>\android-sdk-windows\tools'' to the system search path to enable Android SDK tools to be executed by name no matter what the current directory is.

Simply run adb with the install parameter and the name of the APK to load. If the APK is already installed, you should also specify the -r. Assuming adb’s directory has been added to the search path, you would use:

adb install -r org.me.androidapplication1.apk

'''Note''': If you have an emulator running and a physical device attached to the PC, you can use the -d and -e switches to specify either the device or the emulator, respectively, for example:

adb -e install -r org.me.androidapplication1.apk

[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-Android_Emulator_-_Deployed.png)

Once the APK is installed, you can run it just like you would any other app.

[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-5554Oxygene_JellyBean-Running.png)

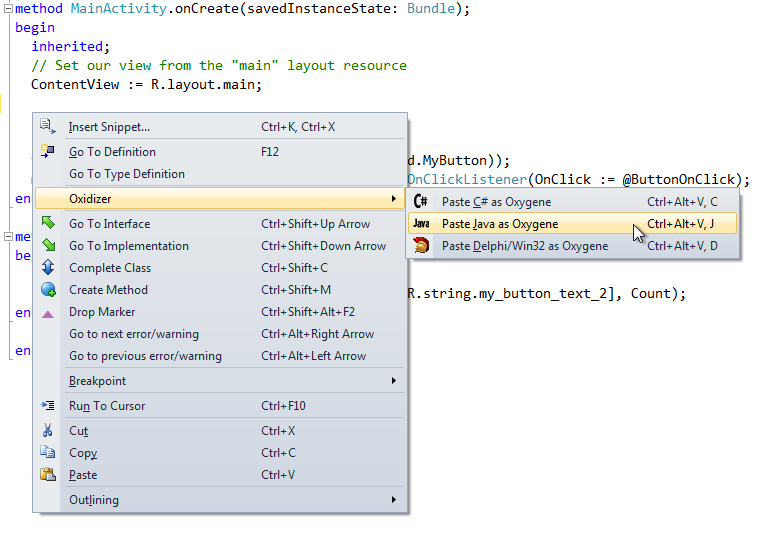
**UI Elements**

The <LinearLayout> tag in ''main.layout-xml'' is one of several layout options. Some of the other layouts are FrameLayout, TableLayout and RelativeLayout.

* [FrameLayout](http://developer.android.com/reference/android/widget/FrameLayout.html) - A frame layout is designed to block out an area on the screen to display a single item.
* [LinearLayout](http://developer.android.com/reference/android/widget/LinearLayout.html) - A layout that arranges its children in a single column (vertical) or a single row (horizontal). The default orientation is horizontal.
* [RelativeLayout](http://developer.android.com/reference/android/widget/RelativeLayout.html) - A layout where the positions of the children can be described in relation to each other or to the parent.
* [TableLayout](http://developer.android.com/reference/android/widget/TableLayout.html) - A layout that arranges its children into rows and columns. A TableLayout consists of a number of [TableRow](http://developer.android.com/reference/android/widget/TableRow.html) objects, each defining a row. TableLayout containers do not display border lines for their rows, columns, or cells. Each row has zero or more cells; each cell can hold one [View](http://developer.android.com/reference/android/view/View.html) object. The table has as many columns as the row with the most cells. A table can leave cells empty. Cells can span columns, as they can in HTML.

Once you have the layout, you can add other controls like [Button](http://developer.android.com/reference/android/widget/Button.html), [ImageButton](http://developer.android.com/reference/android/widget/ImageButton.html), [TextView](http://developer.android.com/reference/android/widget/TextView.html), [ImageView](http://developer.android.com/reference/android/widget/ImageView.html) and others.

There is a [pretty good UI tutorial](http://mobile.tutsplus.com/tutorials/android/android-layout) available. Keep in mind that the code samples will be in Java instead of Oxygene. However, you should find that the Oxidizer IDE feature can help in porting Java code over to the Oxygene syntax. If you have a snippet of Java code in the clipboard, you can press Alt+Ctrl+V, then J and the Oxidizer will do its utmost to translate the code for you. It won’t necessarily do a perfect job as it’s working without context, but it should do the main bulk of the translation for you.

[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-Oxidizer_Java_as_OxygeneAndroid.png)

**Now for some Toast**

When you want to popup a message to your users from an Android app, you make toast. A toast is a small popup window that shows up in front of the current activity for a short time. Simply call the [makeText](http://developer.android.com/reference/android/widget/Toast.html#makeText) method on the [Toast](http://developer.android.com/reference/android/widget/Toast.html) class to create a toast object, then call [show()](http://developer.android.com/reference/android/widget/Toast.html#show) to pop it up on-screen. There are two versions of makeText and each takes 3 parameters. Here is a sample call:

Toast.makeText(self, "Hello World! This is a toast.", Toast.LENGTH\_SHORT).show();

The first parameter is the context that the toast relates to. You can either pass your activity (using self if working in the activity’s method) or a reference to the single application object that exists in the app (accessible via the activity’s [getApplicationContext()](http://developer.android.com/reference/android/content/ContextWrapper.html#getApplicationContext) method), since both [Activity](http://developer.android.com/reference/android/app/Activity.html) and [Application](http://developer.android.com/reference/android/app/Application.html) inherit from the [Context](http://developer.android.com/reference/android/content/Context.html) class.

The second parameter is the text string to display. This can either be a literal string as above or, preferably, a string resource ID, depending on which version of makeText you are using. So to pass a resource ID, you would reference a string defined in ''strings.android-xml'' by using String[R.string.some\_identifier].

The last parameter is the duration before the toast is auto-dismissed, either [Toast.LENGTH\_SHORT](http://developer.android.com/reference/android/widget/Toast.html#LENGTH_SHORT) or [Toast.LENGTH\_LONG](http://developer.android.com/reference/android/widget/Toast.html#LENGTH_LONG).

The call to makeText returns a Toast object. There are other methods on the Toast object to configure it. The show method ultimately displays the toast.

Just add the code above to the ButtonOnClick event handler in MainActivity and when you run it in the emulator or on a device, you will see the toast appear when you click the button.

[](https://docs.elementscompiler.com/Tutorials/FirstApp/VisualStudio/Android/AndroidPrimer-Android_Emulator_-_Running_-_Toast.png)

A toast never receives focus and always disappears after the duration timeout.

**Documentation**

For Android documentation, you can make use of the mass of information provided by Google in their [API Reference](http://d.android.com/reference) and [API Guides](http://d.android.com/guide). Naturally, this documentation is targeted at Java programmers, but that should really be just a minor inconvenience - the method signatures are laid out in [C-style syntax](http://en.wikipedia.org/wiki/C_syntax#Functions) rather than Pascal-style. Oxygene can natively access any of the Android SDK class and call any of the available methods, so the best documentation is the documentation written by the platform provider: Google.

In this primer, various different class types and methods have been linked through to the corresponding Android SDK documentation. You should become familiar with using the Android SDK documentation to learn about how to program Android applications in Oxygene for Java.

**Notes**

One important thing to remember for Android as you start building larger projects and making use of additional libraries is that if you don’t set the “Copy Local” property to true for a referenced library, the compiler presumes the library is available on the system and does not include it in the generated archive. If you set it, it becomes part of the application. For the main platform library, such as android-14, “Copy Local” must be left as false, but for all other libraries it must be set to true to ensure the referenced code will be available when the app runs on an Android device.

# How to make android apps

In this page, you will know how to create the simple hello android application. We are creating the simple example of android using the Eclipse IDE. For creating the simple example:

1. Create the new android project
2. Write the message (optional)
3. Run the android application

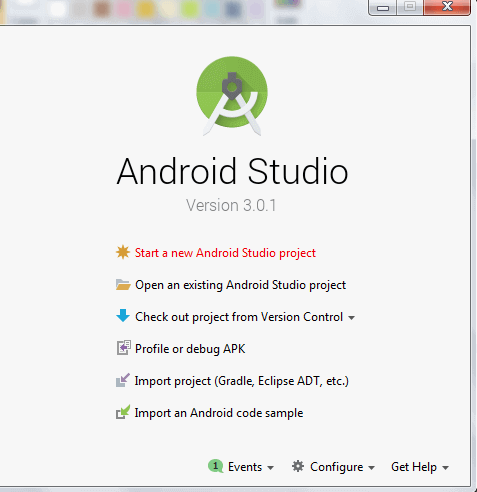
### Hello Android Example

You need to follow the 3 steps mentioned above for creating the Hello android application.

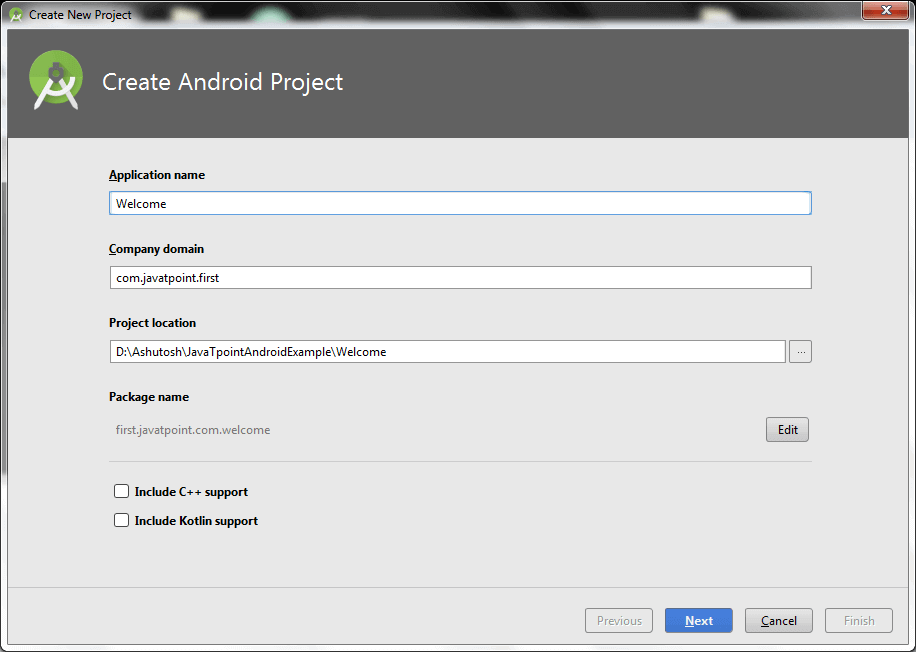
### 1) Create the New Android project

For creating the new android studio project:

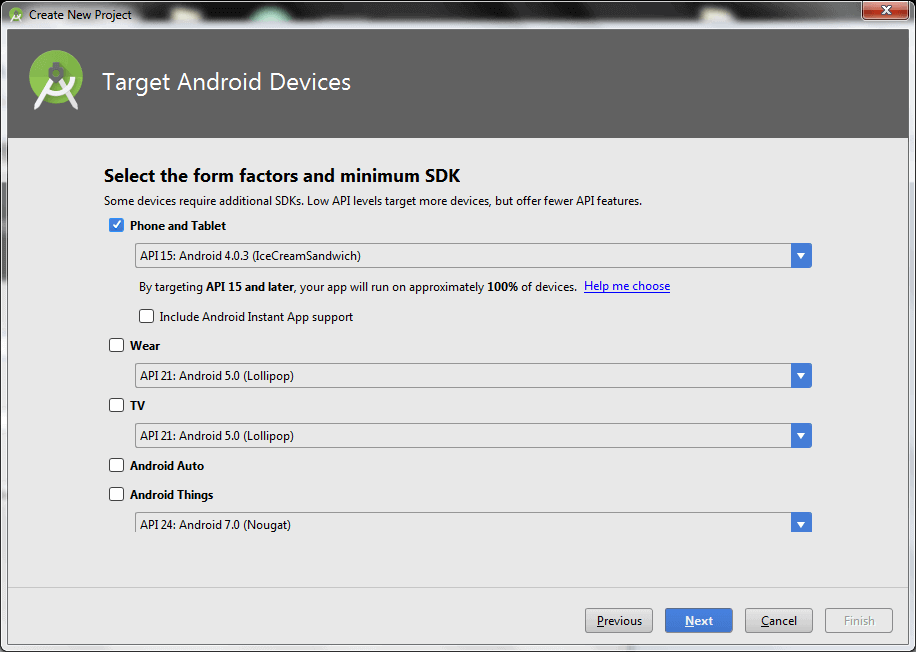
1) Select Start a new Android Studio project



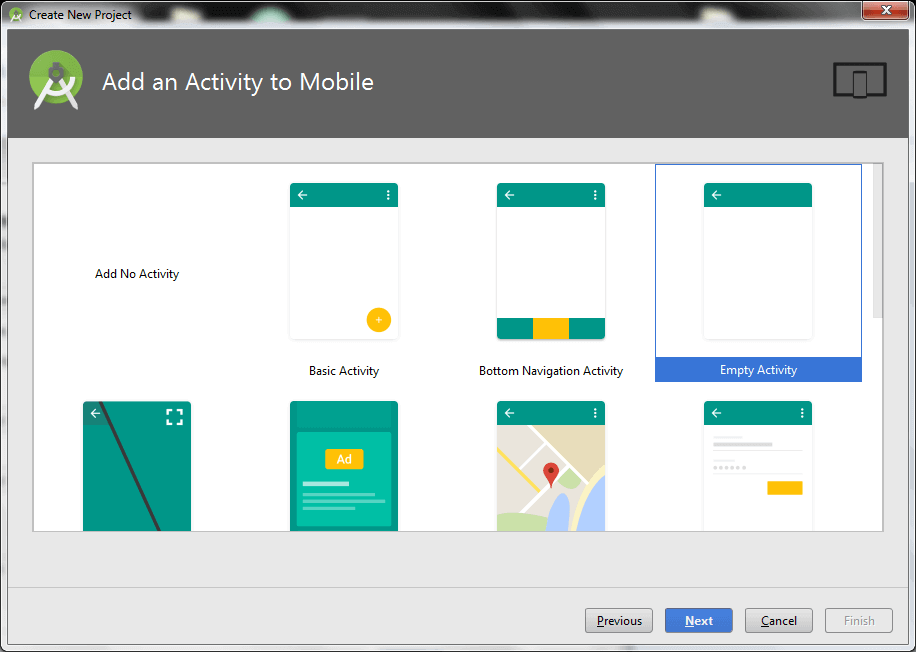
2) Provide the following information: Application name, Company domain, Project location and Package name of application and click next.



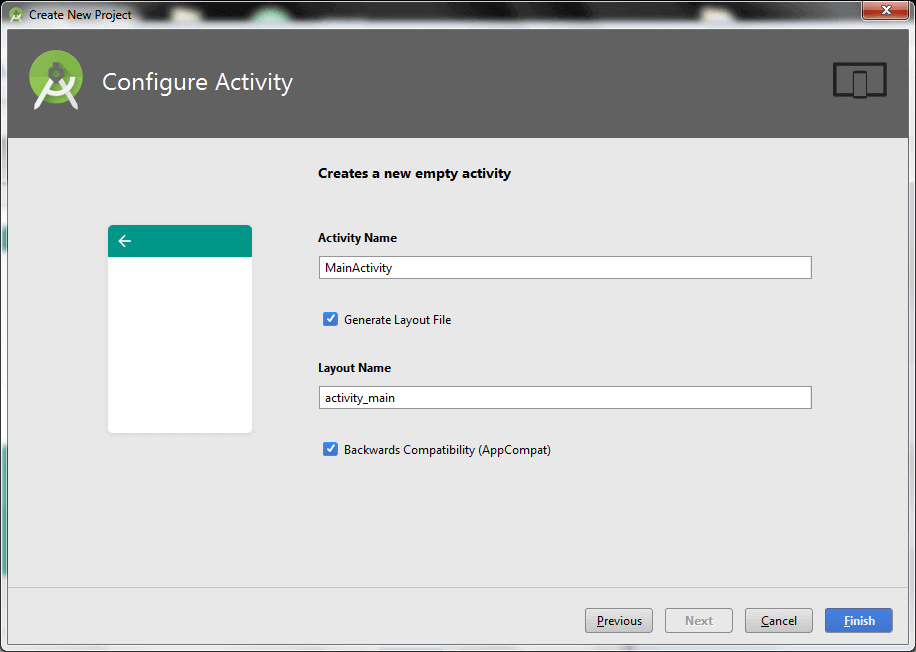
3) Select the API level of application and click next.



4) Select the Activity type (Empty Activity).

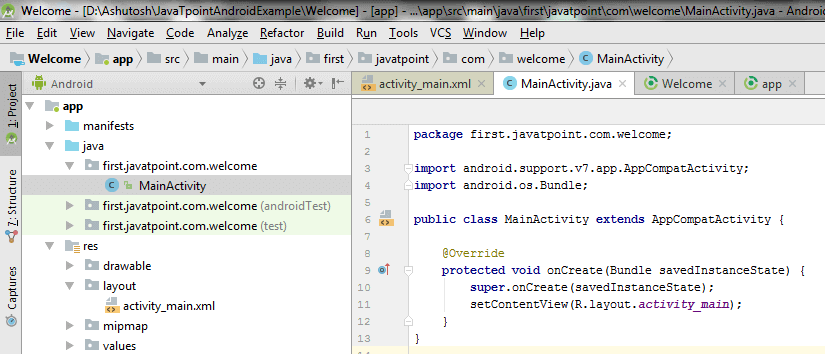


5) Provide the Activity Name and click finish.



After finishing the Activity configuration, Android Studio auto generates the activity class and other required configuration files.

Now an android project has been created. You can explore the android project and see the simple program, it looks like this:



### 2) Write the message

File: activity\_main.xml

Android studio auto generates code for activity\_main.xml file. You may edit this file according to your requirement.

1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3. xmlns:app="http://schemas.android.com/apk/res-auto"
4. xmlns:tools="http://schemas.android.com/tools"
5. android:layout\_width="match\_parent"
6. android:layout\_height="match\_parent"
7. tools:context="first.javatpoint.com.welcome.MainActivity">
9. <TextView
10. android:layout\_width="wrap\_content"
11. android:layout\_height="wrap\_content"
12. android:text="Hello Android!"
13. app:layout\_constraintBottom\_toBottomOf="parent"
14. app:layout\_constraintLeft\_toLeftOf="parent"
15. app:layout\_constraintRight\_toRightOf="parent"
16. app:layout\_constraintTop\_toTopOf="parent" />
18. </android.support.constraint.ConstraintLayout>
19. }

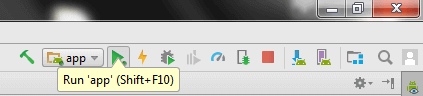
File: MainActivity.java

1. package first.javatpoint.com.welcome;
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
6. public class MainActivity extends AppCompatActivity {
7. @Override
8. protected void onCreate(Bundle savedInstanceState) {
9. super.onCreate(savedInstanceState);
10. setContentView(R.layout.activity\_main);
11. }
12. }

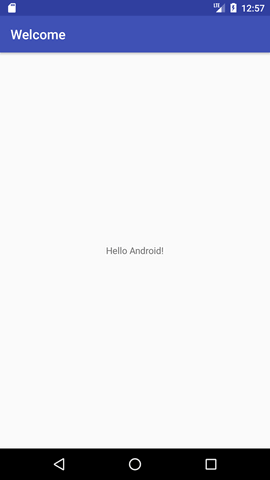
#### To understand the first android application, visit the next page (internal details of hello android example).

## 3) Run the android application

To run the android application, click the run icon on the toolbar or simply press Shift + F10.



The android emulator might take 2 or 3 minutes to boot. So please have patience. After booting the emulator, the android studio installs the application and launches the activity. You will see something like this:



# AndroidManifest.xml file in android

The **AndroidManifest.xml file** contains information of your package, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

* It is **responsible to protect the application** to access any protected parts by providing the permissions.
* It also **declares the android api** that the application is going to use.
* It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

A simple AndroidManifest.xml file looks like this:

1. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2. package="com.javatpoint.hello"
3. android:versionCode="1"
4. android:versionName="1.0" >
6. <uses-sdk
7. android:minSdkVersion="8"
8. android:targetSdkVersion="15" />
10. <application
11. android:icon="@drawable/ic\_launcher"
12. android:label="@string/app\_name"
13. android:theme="@style/AppTheme" >
14. <activity
15. android:name=".MainActivity"
16. android:label="@string/title\_activity\_main" >
17. <intent-filter>
18. <action android:name="android.intent.action.MAIN" />
20. <category android:name="android.intent.category.LAUNCHER" />
21. </intent-filter>
22. </activity>
23. </application>
25. </manifest>

## Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

#### <manifest>

**manifest** is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

#### <application>

**application** is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

**android:icon** represents the icon for all the android application components.

**android:label** works as the default label for all the application components.

**android:theme** represents a common theme for all the android activities.

#### <activity>

**activity** is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

**android:label** represents a label i.e. displayed on the screen.

**android:name** represents a name for the activity class. It is required attribute.

#### <intent-filter>

**intent-filter** is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

#### <action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

#### <category>

It adds a category name to an intent-filter.

# Android Widgets

There are given a lot of **android widgets** with simplified examples such as Button, EditText, AutoCompleteTextView, ToggleButton, DatePicker, TimePicker, ProgressBar etc.

Android widgets are easy to learn. The widely used android widgets with examples are given below:

[Android Button](https://www.javatpoint.com/android-working-with-button)

Let's learn how to perform event handling on button click.

[Android Toast](https://www.javatpoint.com/android-toast-example)

Displays information for the short duration of time.

[Custom Toast](https://www.javatpoint.com/android-custom-toast-example)

We are able to customize the toast, such as we can display image on the toast

[ToggleButton](https://www.javatpoint.com/android-togglebutton-example)

It has two states ON/OFF.

[CheckBox](https://www.javatpoint.com/android-checkbox-example)

Let's see the application of simple food ordering.

[AlertDialog](https://www.javatpoint.com/android-alert-dialog-example)

AlertDialog displays a alert dialog containing the message with OK and Cancel buttons.

[Spinner](https://www.javatpoint.com/android-spinner-example)

Spinner displays the multiple options, but only one can be selected at a time.

[AutoCompleteTextView](https://www.javatpoint.com/android-autocompletetextview-example)

Let's see the simple example of AutoCompleteTextView.

[RatingBar](https://www.javatpoint.com/android-rating-bar-example)

RatingBar displays the rating bar.

[DatePicker](https://www.javatpoint.com/android-datepicker-example)

Datepicker displays the datepicker dialog that can be used to pick the date.

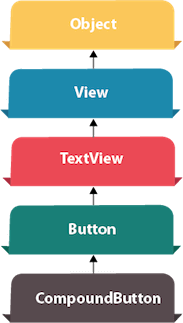
[TimePicker](https://www.javatpoint.com/android-timepicker-example)

TimePicker displays the timepicker dialog that can be used to pick the time.

[ProgressBar](https://www.javatpoint.com/android-progressbar-example)

ProgressBar displays progress task.

# Android Button Example



Android Button represents a push-button. The android.widget.Button is subclass of TextView class and CompoundButton is the subclass of Button class.

There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc.

## Android Button Example with Listener

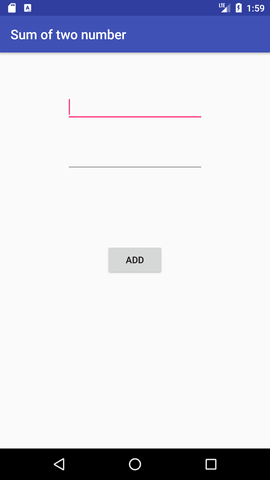
Here, we are going to create two textfields and one button for sum of two numbers. If user clicks button, sum of two input values is displayed on the Toast.

We can perform action on button using different types such as calling listener on button or adding onClick property of button in activity's xml file.

1. button.setOnClickListener(new View.OnClickListener() {
2. @Override
3. public void onClick(View view) {
4. //code
5. }
6. });
7. <Button
8. android:onClick="methodName"
9. />

### Drag the component or write the code for UI in activity\_main.xml

First of all, drag 2 textfields from the Text Fields palette and one button from the Form Widgets palette as shown in the following figure.



The generated code for the ui components will be like this:

File: activity\_main.xml

1. <?xml version="1.0" encoding="utf-8"?>
2. <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3. xmlns:app="http://schemas.android.com/apk/res-auto"
4. xmlns:tools="http://schemas.android.com/tools"
5. android:layout\_width="match\_parent"
6. android:layout\_height="match\_parent"
7. tools:context="example.javatpoint.com.sumoftwonumber.MainActivity">
9. <EditText
10. android:id="@+id/editText1"
11. android:layout\_width="wrap\_content"
12. android:layout\_height="wrap\_content"
13. android:layout\_alignParentTop="true"
14. android:layout\_centerHorizontal="true"
15. android:layout\_marginTop="61dp"
16. android:ems="10"
17. android:inputType="number"
18. tools:layout\_editor\_absoluteX="84dp"
19. tools:layout\_editor\_absoluteY="53dp" />
21. <EditText
22. android:id="@+id/editText2"
23. android:layout\_width="wrap\_content"
24. android:layout\_height="wrap\_content"
25. android:layout\_below="@+id/editText1"
26. android:layout\_centerHorizontal="true"
27. android:layout\_marginTop="32dp"
28. android:ems="10"
29. android:inputType="number"
30. tools:layout\_editor\_absoluteX="84dp"
31. tools:layout\_editor\_absoluteY="127dp" />
33. <Button
34. android:id="@+id/button"
35. android:layout\_width="wrap\_content"
36. android:layout\_height="wrap\_content"
37. android:layout\_below="@+id/editText2"
38. android:layout\_centerHorizontal="true"
39. android:layout\_marginTop="109dp"
40. android:text="ADD"
41. tools:layout\_editor\_absoluteX="148dp"
42. tools:layout\_editor\_absoluteY="266dp" />
43. </RelativeLayout>

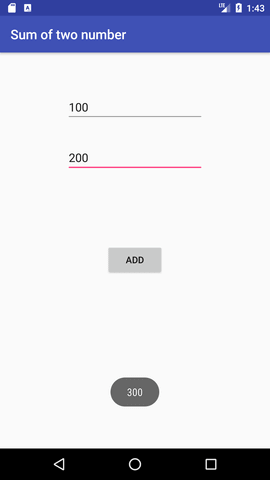
### Activity class

Now write the code to display the sum of two numbers.

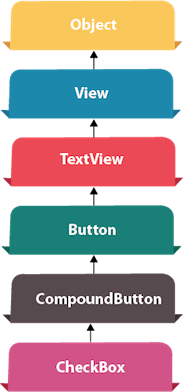
File: MainActivity.java

1. package example.javatpoint.com.sumoftwonumber;
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5. import android.view.View;
6. import android.widget.Button;
7. import android.widget.EditText;
8. import android.widget.Toast;
10. public class MainActivity extends AppCompatActivity {
11. private EditText edittext1, edittext2;
12. private Button buttonSum;
14. @Override
15. protected void onCreate(Bundle savedInstanceState) {
16. super.onCreate(savedInstanceState);
17. setContentView(R.layout.activity\_main);
19. addListenerOnButton();
20. }
22. public void addListenerOnButton() {
23. edittext1 = (EditText) findViewById(R.id.editText1);
24. edittext2 = (EditText) findViewById(R.id.editText2);
25. buttonSum = (Button) findViewById(R.id.button);
27. buttonSum.setOnClickListener(new View.OnClickListener() {
28. @Override
29. public void onClick(View view) {
30. String value1=edittext1.getText().toString();
31. String value2=edittext2.getText().toString();
32. int a=Integer.parseInt(value1);
33. int b=Integer.parseInt(value2);
34. int sum=a+b;
35. Toast.makeText(getApplicationContext(),String.valueOf(sum), Toast.LENGTH\_LONG).show();
36. }
37. });
38. }
39. }

#### Output:



# Android CheckBox Example



**Android CheckBox** is a type of two state button either checked or unchecked.

There can be a lot of usage of checkboxes. For example, it can be used to know the hobby of the user, activate/deactivate the specific action etc.

Android CheckBox class is the subclass of CompoundButton class.

## Android CheckBox class

The android.widget.CheckBox class provides the facility of creating the CheckBoxes.

#### Methods of CheckBox class

There are many inherited methods of View, TextView, and Button classes in the CheckBox class. Some of them are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| public boolean isChecked() | Returns true if it is checked otherwise false. |
| public void setChecked(boolean status) | Changes the state of the CheckBox. |

## Android CheckBox Example

#### activity\_main.xml

Drag the three checkboxes and one button for the layout. Now the activity\_main.xml file will look like this:

File: activity\_main.xml

1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3. xmlns:app="http://schemas.android.com/apk/res-auto"
4. xmlns:tools="http://schemas.android.com/tools"
5. android:layout\_width="match\_parent"
6. android:layout\_height="match\_parent"
7. tools:context="example.javatpoint.com.checkbox.MainActivity">

10. <CheckBox
11. android:id="@+id/checkBox"
12. android:layout\_width="wrap\_content"
13. android:layout\_height="wrap\_content"
14. android:layout\_marginLeft="144dp"
15. android:layout\_marginTop="68dp"
16. android:text="Pizza"
17. app:layout\_constraintStart\_toStartOf="parent"
18. app:layout\_constraintTop\_toTopOf="parent" />
20. <CheckBox
21. android:id="@+id/checkBox2"
22. android:layout\_width="wrap\_content"
23. android:layout\_height="wrap\_content"
24. android:layout\_marginLeft="144dp"
25. android:layout\_marginTop="28dp"
26. android:text="Coffee"
27. app:layout\_constraintStart\_toStartOf="parent"
28. app:layout\_constraintTop\_toBottomOf="@+id/checkBox" />
30. <CheckBox
31. android:id="@+id/checkBox3"
32. android:layout\_width="wrap\_content"
33. android:layout\_height="wrap\_content"
34. android:layout\_marginLeft="144dp"
35. android:layout\_marginTop="28dp"
36. android:text="Burger"
37. app:layout\_constraintStart\_toStartOf="parent"
38. app:layout\_constraintTop\_toBottomOf="@+id/checkBox2" />
40. <Button
41. android:id="@+id/button"
42. android:layout\_width="wrap\_content"
43. android:layout\_height="wrap\_content"
44. android:layout\_marginLeft="144dp"
45. android:layout\_marginTop="184dp"
46. android:text="Order"
47. app:layout\_constraintStart\_toStartOf="parent"
48. app:layout\_constraintTop\_toBottomOf="@+id/checkBox3" />
50. </android.support.constraint.ConstraintLayout>

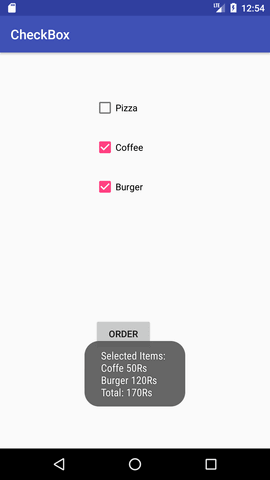
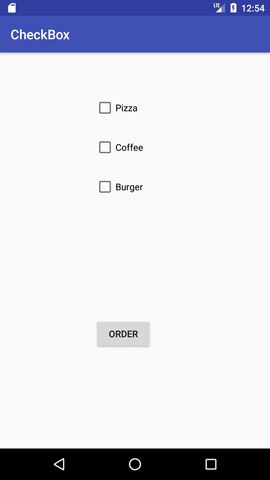
#### Activity class

Let's write the code to check which toggle button is ON/OFF.

File: MainActivity.java

1. package example.javatpoint.com.checkbox;
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5. import android.view.View;
6. import android.widget.Button;
7. import android.widget.CheckBox;
8. import android.widget.Toast;
10. public class MainActivity extends AppCompatActivity {
11. CheckBox pizza,coffe,burger;
12. Button buttonOrder;
13. @Override
14. protected void onCreate(Bundle savedInstanceState) {
15. super.onCreate(savedInstanceState);
16. setContentView(R.layout.activity\_main);
17. addListenerOnButtonClick();
18. }
19. public void addListenerOnButtonClick(){
20. //Getting instance of CheckBoxes and Button from the activty\_main.xml file
21. pizza=(CheckBox)findViewById(R.id.checkBox);
22. coffe=(CheckBox)findViewById(R.id.checkBox2);
23. burger=(CheckBox)findViewById(R.id.checkBox3);
24. buttonOrder=(Button)findViewById(R.id.button);
26. //Applying the Listener on the Button click
27. buttonOrder.setOnClickListener(new View.OnClickListener(){
29. @Override
30. public void onClick(View view) {
31. int totalamount=0;
32. StringBuilder result=new StringBuilder();
33. result.append("Selected Items:");
34. if(pizza.isChecked()){
35. result.append("\nPizza 100Rs");
36. totalamount+=100;
37. }
38. if(coffe.isChecked()){
39. result.append("\nCoffe 50Rs");
40. totalamount+=50;
41. }
42. if(burger.isChecked()){
43. result.append("\nBurger 120Rs");
44. totalamount+=120;
45. }
46. result.append("\nTotal: "+totalamount+"Rs");
47. //Displaying the message on the toast
48. Toast.makeText(getApplicationContext(), result.toString(), Toast.LENGTH\_LONG).show();
49. }
51. });
52. }
53. }

#### Output:



# Android RadioButton

**RadioButton** is a two states button which is either checked or unchecked. If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user.

RadioButton is generally used with RadioGroup. RadioGroup contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.

## Example of Radio Button

In this example, we are going to implement single radio button separately as well as radio button in **RadioGroup**.

### activity\_main.xml

**File: activity\_main.xml**

1. <?xml version="1.0" encoding="utf-8"?>
2. <LinearLayout
3. xmlns:android="http://schemas.android.com/apk/res/android"
4. xmlns:tools="http://schemas.android.com/tools"
5. android:layout\_width="match\_parent"
6. android:layout\_height="match\_parent"
7. android:orientation="vertical"
8. tools:context="example.javatpoint.com.radiobutton.MainActivity">
10. <TextView
11. android:id="@+id/textView1"
12. android:layout\_width="fill\_parent"
13. android:layout\_height="wrap\_content"
14. android:layout\_marginTop="30dp"
15. android:gravity="center\_horizontal"
16. android:textSize="22dp"
17. android:text="Single Radio Buttons" />


21. <!--   Default RadioButtons  -->
23. <RadioButton
24. android:id="@+id/radioButton1"
25. android:layout\_width="fill\_parent"
26. android:layout\_height="wrap\_content"
27. android:layout\_gravity="center\_horizontal"
28. android:text="Radio Button 1"
29. android:layout\_marginTop="20dp"
31. android:textSize="20dp" />
32. <RadioButton
33. android:id="@+id/radioButton2"
34. android:layout\_width="fill\_parent"
35. android:layout\_height="wrap\_content"
36. android:text="Radio Button 2"
37. android:layout\_marginTop="10dp"
39. android:textSize="20dp" />

42. <View
43. android:layout\_width="fill\_parent"
44. android:layout\_height="1dp"
45. android:layout\_marginTop="20dp"
46. android:background="#B8B894" />
48. <TextView
49. android:id="@+id/textView2"
50. android:layout\_width="fill\_parent"
51. android:layout\_height="wrap\_content"
52. android:layout\_marginTop="30dp"
53. android:gravity="center\_horizontal"
54. android:textSize="22dp"
55. android:text="Radio button inside RadioGroup" />

58. <!--   Customized RadioButtons  -->

61. <RadioGroup
62. android:layout\_width="wrap\_content"
63. android:layout\_height="wrap\_content"
64. android:id="@+id/radioGroup">
66. <RadioButton
67. android:id="@+id/radioMale"
68. android:layout\_width="fill\_parent"
69. android:layout\_height="wrap\_content"
70. android:text="  Male"
71. android:layout\_marginTop="10dp"
72. android:checked="false"
73. android:textSize="20dp" />
75. <RadioButton
76. android:id="@+id/radioFemale"
77. android:layout\_width="fill\_parent"
78. android:layout\_height="wrap\_content"
79. android:text="   Female"
80. android:layout\_marginTop="20dp"
81. android:checked="false"
83. android:textSize="20dp" />
84. </RadioGroup>
86. <Button
87. android:layout\_width="wrap\_content"
88. android:layout\_height="wrap\_content"
89. android:text="Show Selected"
90. android:id="@+id/button"
91. android:onClick="onclickbuttonMethod"
92. android:layout\_gravity="center\_horizontal" />

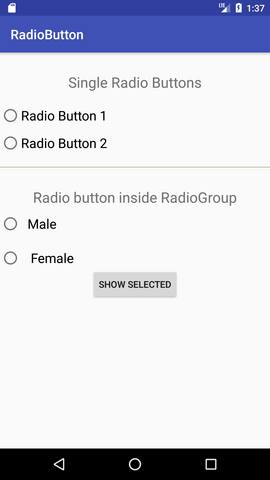
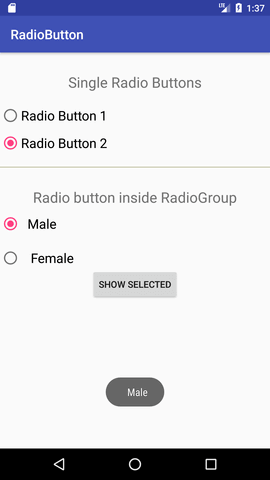
95. </LinearLayout>

### Activity class

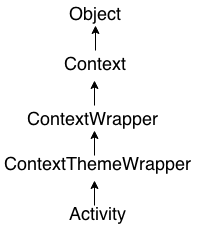
**File: MainActivity.java**

1. package example.javatpoint.com.radiobutton;
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5. import android.view.View;
6. import android.widget.Button;
7. import android.widget.RadioButton;
8. import android.widget.RadioGroup;
9. import android.widget.Toast;
11. public class MainActivity extends AppCompatActivity {
12. Button button;
13. RadioButton genderradioButton;
14. RadioGroup radioGroup;
15. @Override
16. protected void onCreate(Bundle savedInstanceState) {
17. super.onCreate(savedInstanceState);
18. setContentView(R.layout.activity\_main);
19. radioGroup=(RadioGroup)findViewById(R.id.radioGroup);
20. }
21. public void onclickbuttonMethod(View v){
22. int selectedId = radioGroup.getCheckedRadioButtonId();
23. genderradioButton = (RadioButton) findViewById(selectedId);
24. if(selectedId==-1){
25. Toast.makeText(MainActivity.this,"Nothing selected", Toast.LENGTH\_SHORT).show();
26. }
27. else{
28. Toast.makeText(MainActivity.this,genderradioButton.getText(), Toast.LENGTH\_SHORT).show();
29. }
31. }
32. }

Output

# Android Activity Lifecycle



**Android Activity Lifecycle** is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

An activity is the single screen in android. It is like window or frame of Java.

By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different states.

## Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **onCreate** | called when activity is first created. |
| **onStart** | called when activity is becoming visible to the user. |
| **onResume** | called when activity will start interacting with the user. |
| **onPause** | called when activity is not visible to the user. |
| **onStop** | called when activity is no longer visible to the user. |
| **onRestart** | called after your activity is stopped, prior to start. |
| **onDestroy** | called before the activity is destroyed. |



File: activity\_main.xml

1. <?xml version="1.0" encoding="utf-8"?>
2. <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3. xmlns:app="http://schemas.android.com/apk/res-auto"
4. xmlns:tools="http://schemas.android.com/tools"
5. android:layout\_width="match\_parent"
6. android:layout\_height="match\_parent"
7. tools:context="example.javatpoint.com.activitylifecycle.MainActivity">
9. <TextView
10. android:layout\_width="wrap\_content"
11. android:layout\_height="wrap\_content"
12. android:text="Hello World!"
13. app:layout\_constraintBottom\_toBottomOf="parent"
14. app:layout\_constraintLeft\_toLeftOf="parent"
15. app:layout\_constraintRight\_toRightOf="parent"
16. app:layout\_constraintTop\_toTopOf="parent" />
18. </android.support.constraint.ConstraintLayout>

### Android Activity Lifecycle Example

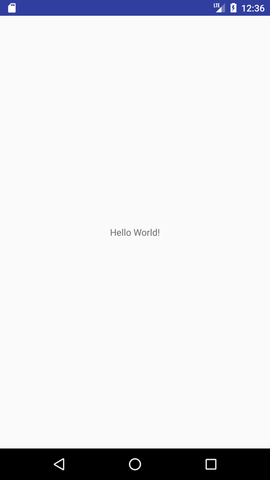
It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

File: MainActivity.java

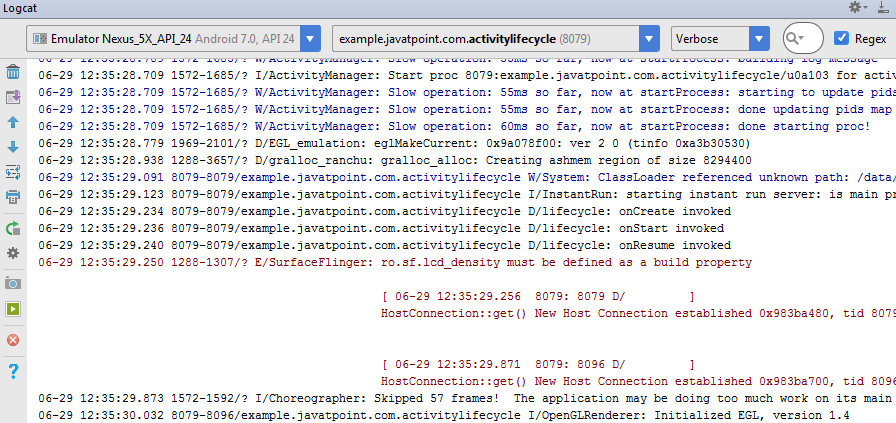
1. package example.javatpoint.com.activitylifecycle;
3. import android.app.Activity;
4. import android.os.Bundle;
5. import android.util.Log;
7. public class MainActivity extends Activity {
9. @Override
10. protected void onCreate(Bundle savedInstanceState) {
11. super.onCreate(savedInstanceState);
12. setContentView(R.layout.activity\_main);
13. Log.d("lifecycle","onCreate invoked");
14. }
15. @Override
16. protected void onStart() {
17. super.onStart();
18. Log.d("lifecycle","onStart invoked");
19. }
20. @Override
21. protected void onResume() {
22. super.onResume();
23. Log.d("lifecycle","onResume invoked");
24. }
25. @Override
26. protected void onPause() {
27. super.onPause();
28. Log.d("lifecycle","onPause invoked");
29. }
30. @Override
31. protected void onStop() {
32. super.onStop();
33. Log.d("lifecycle","onStop invoked");
34. }
35. @Override
36. protected void onRestart() {
37. super.onRestart();
38. Log.d("lifecycle","onRestart invoked");
39. }
40. @Override
41. protected void onDestroy() {
42. super.onDestroy();
43. Log.d("lifecycle","onDestroy invoked");
44. }
45. }

#### Output:

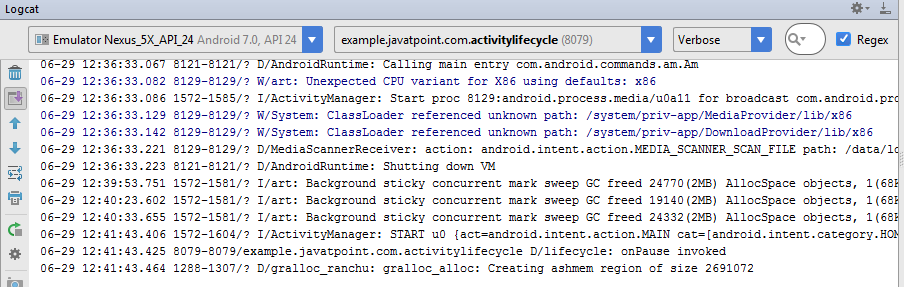
You will not see any output on the emulator or device. You need to open logcat.



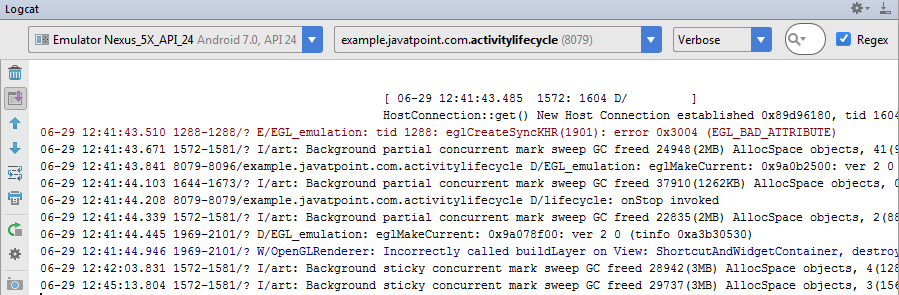
Now see on the logcat: onCreate, onStart and onResume methods are invoked.



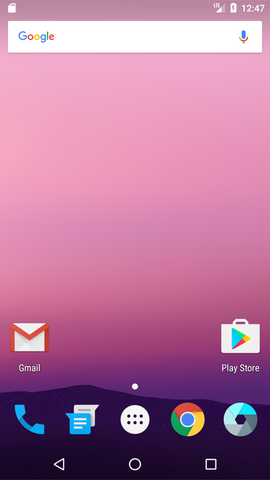
Now click on the HOME Button. You will see onPause method is invoked.



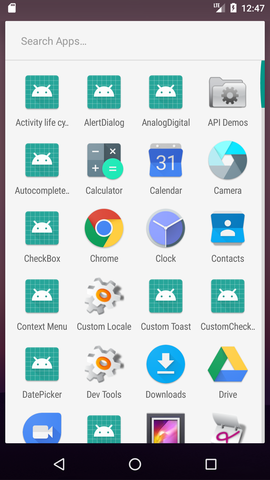
After a while, you will see onStop method is invoked.



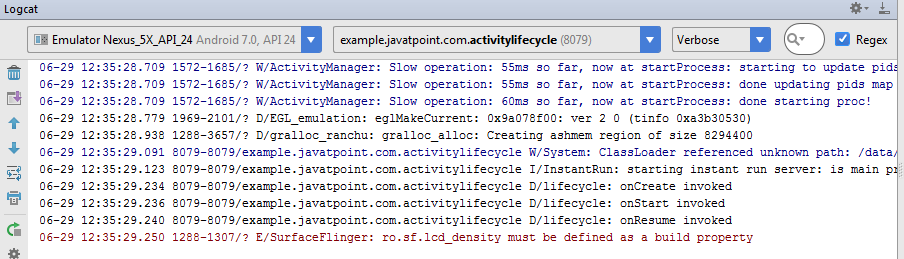
Now see on the emulator. It is on the home. Now click on the center button to launch the app again.



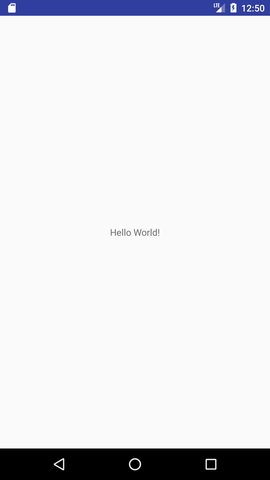
Now click on the lifecycleactivity icon.



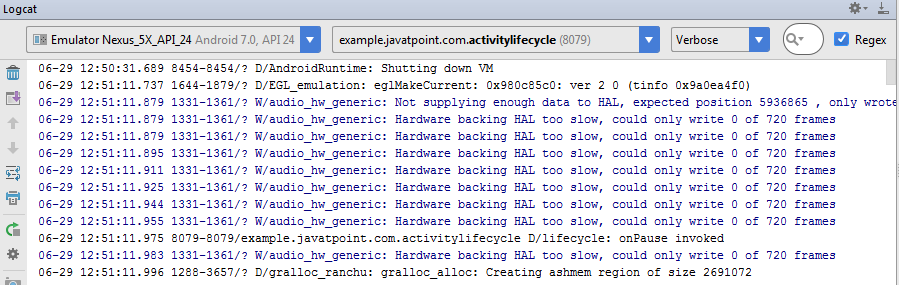
Now see on the logcat: onRestart, onStart and onResume methods are invoked.



If you see the emulator, application is started again.



Now click on the back button. Now you will see onPause methods is invoked.



After a while, you will see onStop and onDestroy methods are invoked.

