Getting Started With Android App Development

Table of Contents

Chapter 1: Android OS

1.1 Introduction to Android Operating System

Android is an **OS** created by Google™ for use on mobile devices, such as smartphones and tablets. It's an **OS** that's available on devices made by a variety of manufacturers, giving you more choices of device style and pricing. Also, with the **Android OS**, you can customize your device in many ways. **Android** uses the **Linux** kernel under the hood. Because **Linux** is open-source, Google's **Android** developers could modify the **Linux** kernel to fit their needs. **Linux** gives the **Android** developers a pre-built, already maintained operating system kernel to start with so they don't have to write their own kernel.

Android, Inc. was founded in **Palo Alto, California** in October 2003 by **Andy Rubin** (co-founder of **Danger,** former employee of **Apple Inc.**), **Rich Miner** (co-founder of Wildfire Communications, Inc.), **Nick Sears** (once Vice President at **T-Mobile**), and **Chris White** (headed design and interface development at **WebTV**) To develop, in Rubin's words,

"*Smarter mobile device that is more aware of its owner's location and preferences*".

The early intentions of the company were to develop an advanced operating system for **digital cameras**. Though, when it was realized that the market for the devices was not large enough, the company diverted its efforts toward producing a smartphone operating system that would rival **Symbian** and Microsoft **Windows Mobile**. Despite the past accomplishments of the founders and early employees, **Android Inc.** operated secretly, revealing only that it was working on software for mobile phones. That same year, Rubin ran out of money. **Steve Perlman**, a close friend of Rubin, brought him **$10,000** in cash in an envelope and refused a stake in the company.

In July 2005, **Google** acquired Android Inc. at **$50 million**, whose key employees, including Rubin, Miner and White, stayed at the company after the acquisition. Not much was known about **Android Inc.** at the time, but many assumed that Google was planning to enter the mobile phone market with this move. At Google, the team led by Rubin developed a mobile device platform powered by the **Linux kernel**. Google marketed the platform to handset makers and **carriers** on the promise of providing a flexible, upgradeable system. Google had lined up a series of hardware component and software partners and signaled to carriers that it was open to various degrees of cooperation on their part.

1.2 Android Versions:

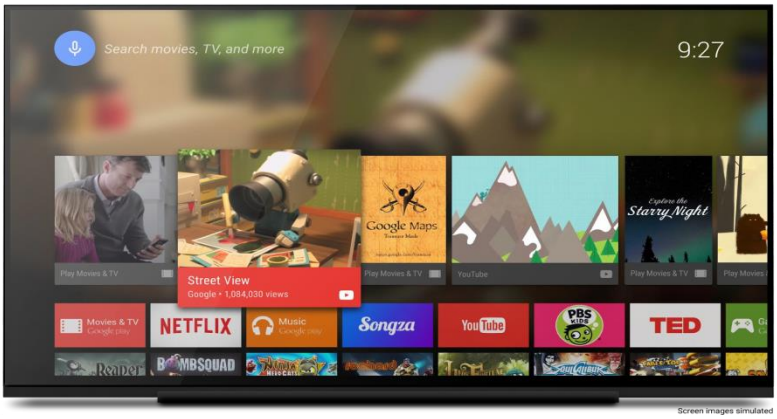
Code name	Version number	Initial release date	API level	Support Status
N/A	1.0	23 September 2008	1	Discontinued
	1.1	9 February 2009	2	Discontinued
Cupcake	1.5	27 April 2009	3	Discontinued
Donut	1.6	15 September 2009	4	Discontinued
Eclair	2.0 – 2.1	26 October 2009	5–7	Discontinued
Froyo	2.2 – 2.2.3	20 May 2010	8	Discontinued
Gingerbread	2.3 – 2.3.7	6 December 2010	9–10	Discontinued
Honeycomb	3.0 – 3.2.6	22 February 2011	11–13	Discontinued
Ice Cream Sandwich	4.0 – 4.0.4	18 October 2011	14–15	Discontinued
Jelly Bean	4.1 – 4.3.1	9 July 2012	16–18	Discontinued
KitKat	4.4 – 4.4.4	31 October 2013	19–20	Security Updates Only
Lollipop	5.0 – 5.1.1	12 November 2014	21–22	Supported
Marshmallow	6.0 – 6.0.1	5 October 2015	23	Supported
Nougat	**7.0 – 7.1**	**22 August 2016**	**24–25**	Supported

Nomenclature is done on the name of desserts

1.3 Devices Supported by Android OS

Mobile Devices

Tablet/Note

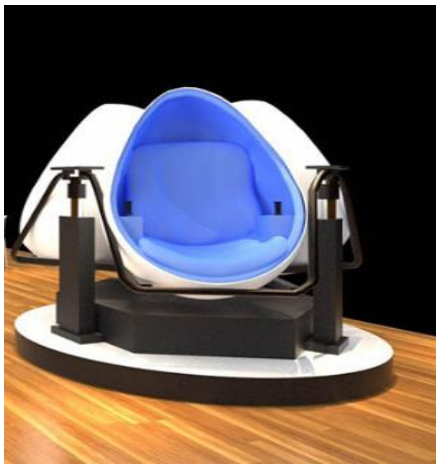


Smart TV

Smart Camera

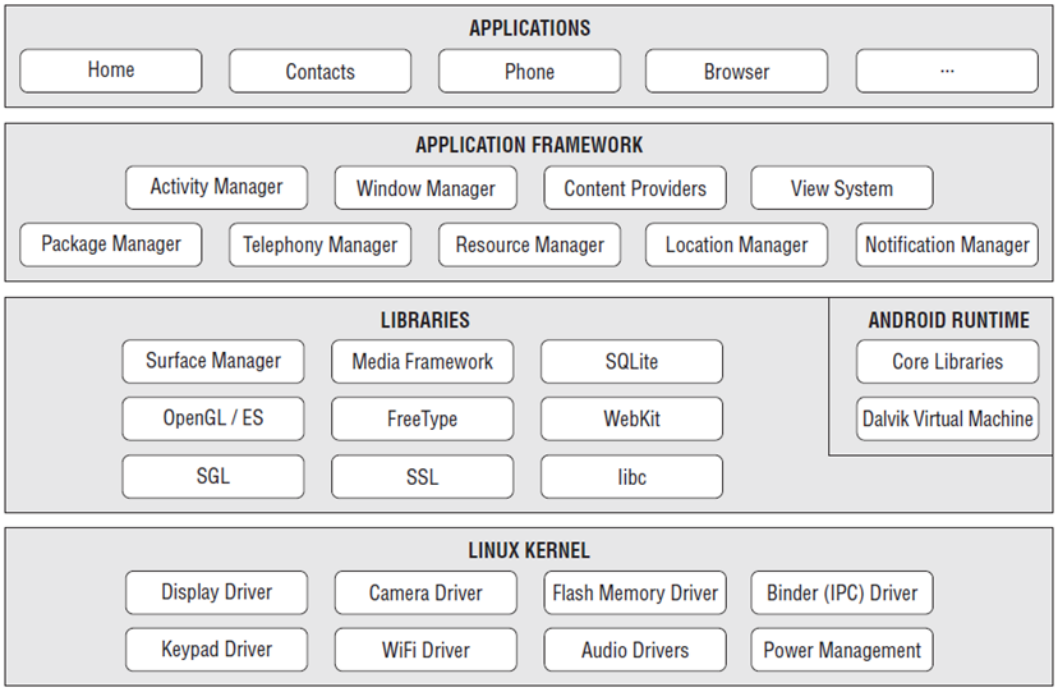
Gear Watch

Gear Virtual Reality headset



Gear Virtual Reality Chair

1.4 Android OS Architecture



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, there 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

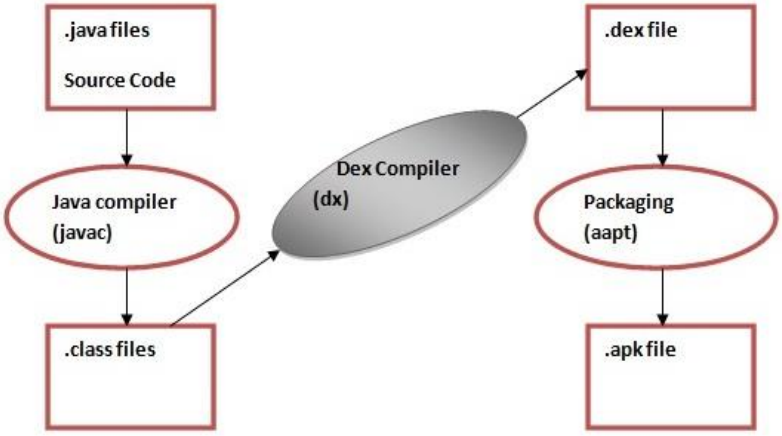
On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

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 kernel.

1.5.1 Dalvik Virtual Machine

As we know the modern JVM is high performance and provides excellent memory management. But it needs to be optimized for low-powered handheld devices as well. The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for *memory*, *battery life* and *performance*. Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file. Let's see the compiling and packaging process from the source file:



DVM runtime execution

Dan Bornstein at Delvik town

The **javac tool** compiles the java source file into the class file. The **dx tool** takes all the class files of your application and generates a single .dex file. It is a platform-specific tool. The **Android Assets Packaging Tool (aapt)** handles the packaging process.

1.5.2 Android Runtime Virtual Machine

ART (Android RunTime) is the **next version** of **Dalvik V M**. **Dalvik** is the runtime, bytecode, and VM used by the Android system for running Android applications.

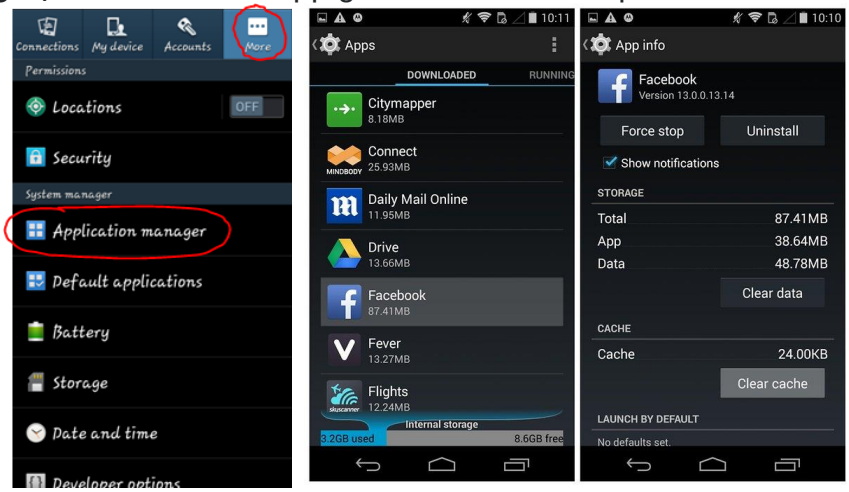
Android RunTime uses different approach. At the time of installation of apk file, ART compiles the apk into machine code which can be run directly by the device (with the help of ART). Although the compilation takes some time, thus the installation duration will increase, which is negligible, unless you're installing a pretty big apk. Now since the compilation is already done, the apk will work as fast as Native C++ code.

ART has two main features compared to **Dalvik V M** :

- **Ahead-of-Time** (**AOT**) compilation, which improves speed (particularly startup time) and reduces memory footprint (no **JIT**)
- **Improved Garbage Collection (GC)**

1.5.3 Dalvik-cache

When you say "*dalvik-cache*", I assume you mean the **/data/dalvik-cache** directory that can be found on typical Android devices. When you install an application on Android, it performs some modifications and optimizations on that application's dex file (the file that contains all the dalvik bytecode for the application). It then caches the resulting odex (*optimized dex*) file in the**/data/dalvik-cache** directory, so that it doesn't have to perform the optimization process every time it loads an application. Cache of the specific App can also be manually cleared by Opening the **Application manager,** then select the app go for clear cache option.

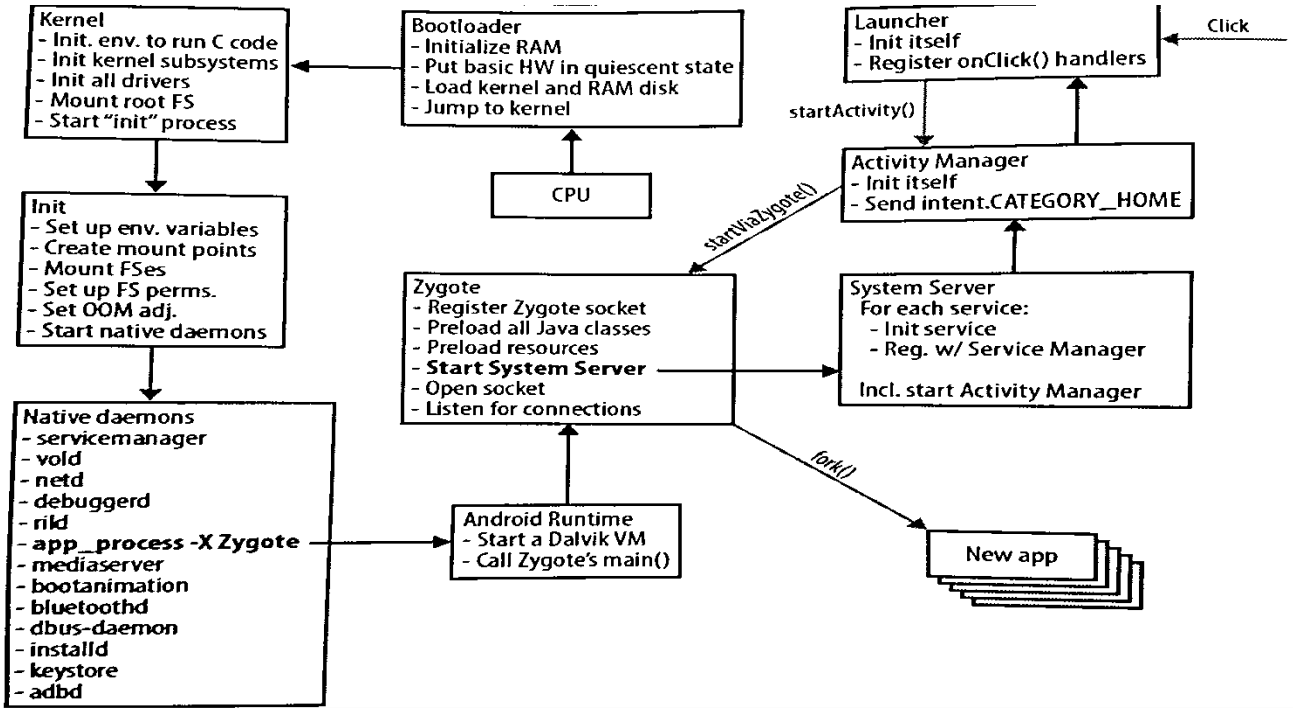


1.6 Zygote Process

Android at its core has a process they call the “Zygote”, which starts up at init(). It gets it's name from dictionary definition: "It is the initial cell formed when a new organism is produced". This process is a “Warmed-up” process, which means it’s a process that’s been initialized and has all the core libraries linked in. When you start an application, the Zygote is forked, so now there are 2 VMs. The real speedup is achieved by NOT copying the shared libraries. This memory will only be copied if the new process tries to modify it. This means that all of the core libraries can exist in a single place because they are read only.

Zygote receives a request to launch an App through /dev/socket/zygote. Once it happens it trigger a **fork()** call. Here is where the virtues of a great design take action. When a process forks, it creates a clone of itself. It replicates itself in another memory space. This is done pretty efficiently. When this happens to Zygote, it creates an exact and clean new Dalvik VM, preloaded with all necessary classes and resources that any App will need. This makes the process of creating a VM and load resources pretty efficiently. But the design goes farther. As we know, Android runs on Linux. The Linux Kernel implements a strategy call **Copy On Write** (COW). What this means is that during the fork process, no memory is actually copy to another space. It is shared and marked as **copy-on-write** which means that when a process attempt to modify that memory, the kernel will intercept the call and do the copy of that piece of memory. In the case of Android those libraries are not writable. This means that all process forked from Zygote are using the exact same copy of the system classes and

resources. Another benefit is the real memory saving. No matter how many applications are started the increase in memory usage will be a lot smaller.



1.7 Termination Process

Q) When does a process die?

Processes can be killed in a couple discrete ways:

- An application can call a method to kill processes it has permission to kill. This means if the process isn't part of the same application, it can't kill other processes. On install you can actually grant an application permission to kill other applications, but this is something you don't typically do.
- The Android OS has a least recently used queue that keeps track of which applications haven't been used. If the OS starts to run out of memory, it will kill the least recently used application. There is also priority given to applications that a user is interacting with, or background services the user is interacting with. A detailed article on these preferences can be found here.

Q) How does a process die?

Obviously as detailed above, a process can be killed using **Process.killProcess (int pid)**, but point 2 is a bit vague. This is done via a Linux driver that is loaded for android only (right now). This driver has been the subject of much debate as the mainstream Linux code should be different.

Chapter 2: Android App Development IDEs

2.1 Eclipse IDE:

It is the traditional way of developing an android App where we need to setup individually JDK, SDK, AVD and ADT. This problem of complication in installation process is solved by Android studio.

JDK: JAVA Development Kit
SDK: Software Development Kit
ADT: Android Development Tool
AVD: Android Virtual Device

2.2 Android Studio

Android Studio is the official **integrated development environment** (IDE) for **Android** platform development.

It was announced on May 16, 2013 at the **Google I/O** conference. Android Studio is freely available under the **Apache License 2.0**.

Android Studio was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0.

Based on **JetBrains, IntelliJ IDEA** software, Android Studio is designed specifically for Android development. It is available for download on **Windows, Mac OS X** and **Linux**, and replaced **Eclipse Android Development Tools** (ADT) as Google's primary IDE for native Android application development.

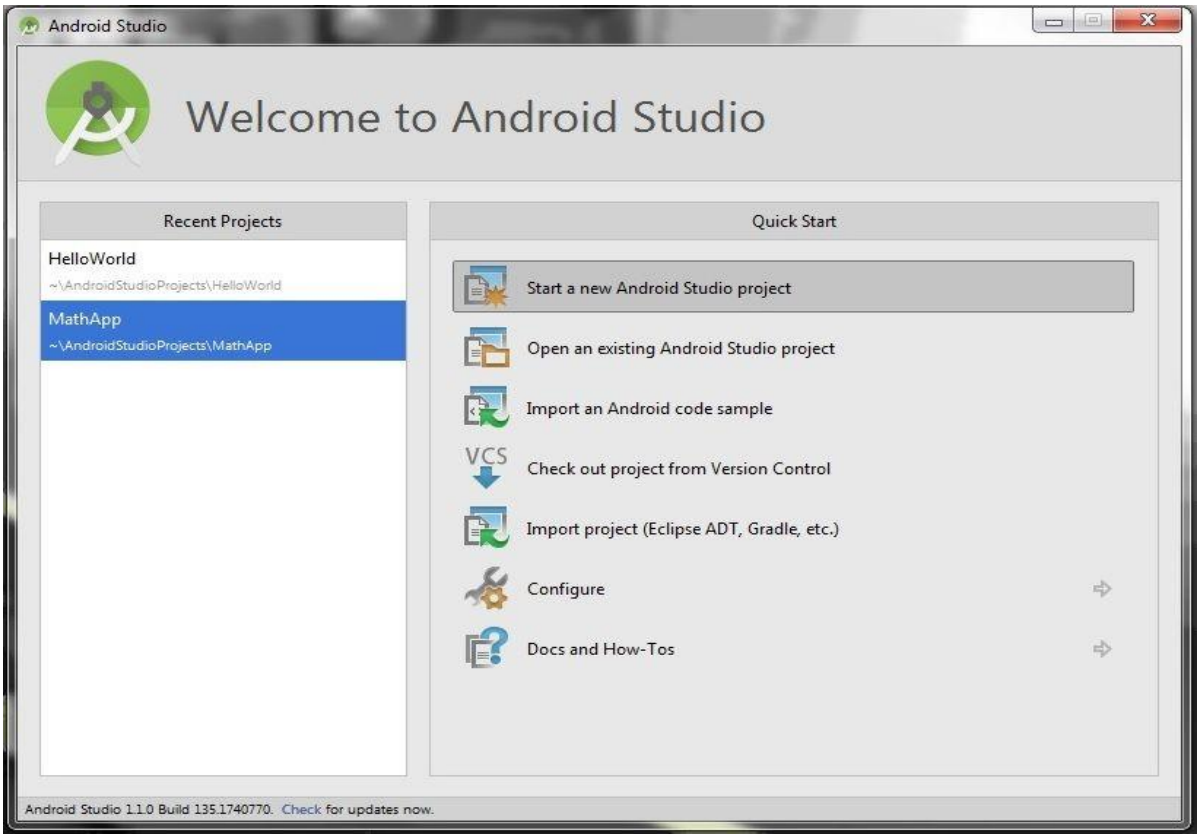
Features

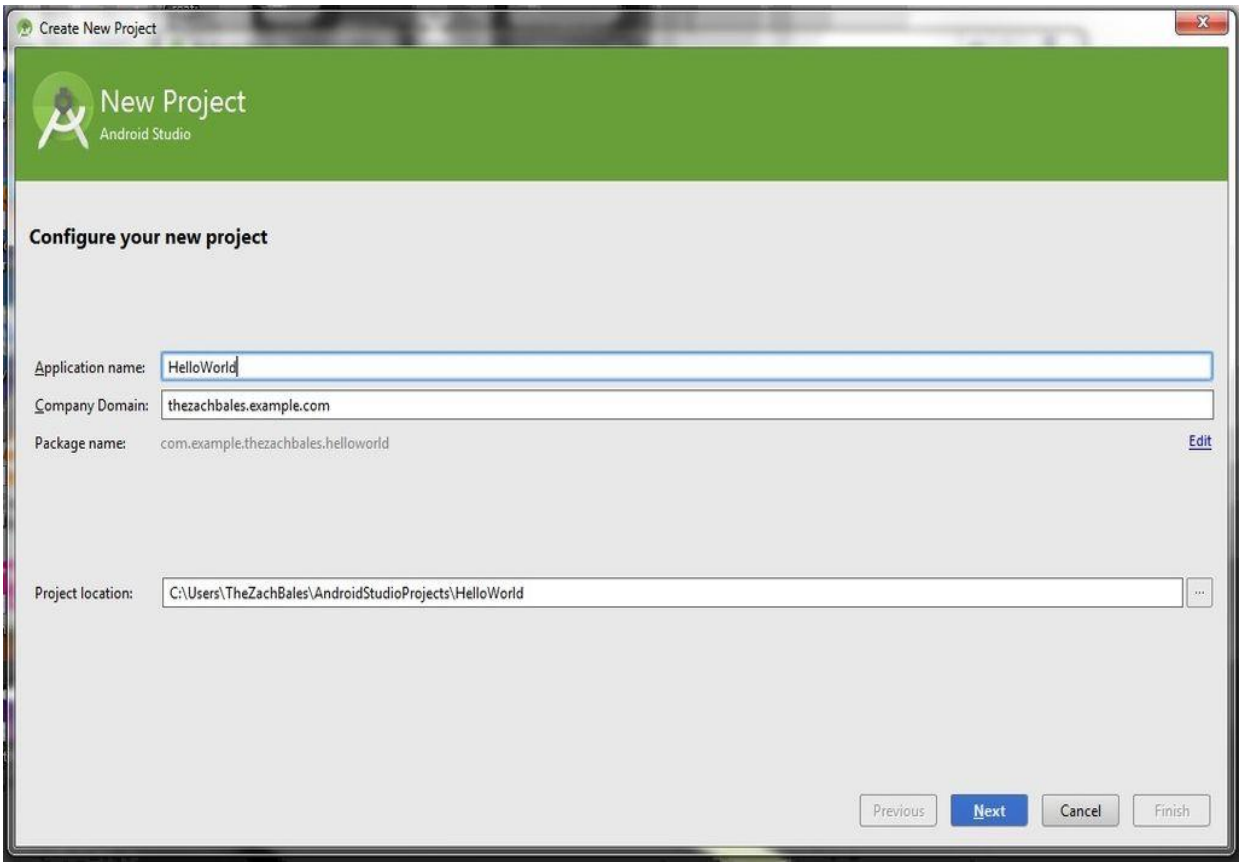
New features are expected to be rolled out with each release of Android Studio. The following features are provided in the current stable version:

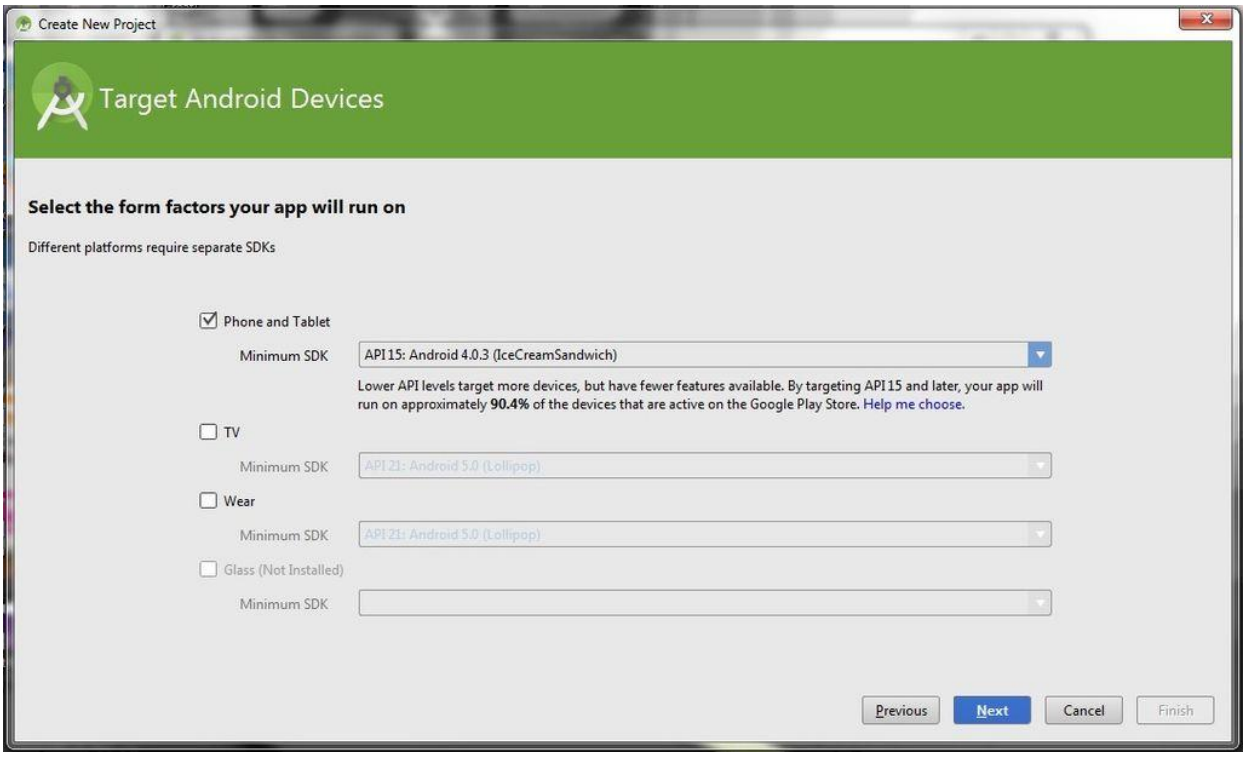
- **Gradle**-based build support.
- Android-specific **refactoring** and quick fixes.
- **Lint** tools to catch performance, usability, version compatibility and other problems.
- **ProGuard** integration and app-signing capabilities.
- Template-based wizards to create common Android designs and components.
- A rich **layout editor** that allows users to drag-and-drop UI components, option to **preview layouts** on multiple screen configurations.
- Support for building **Android Wear** apps
- Built-in support for Google Cloud Platform, enabling integration with Google Cloud Messaging and App Engine.

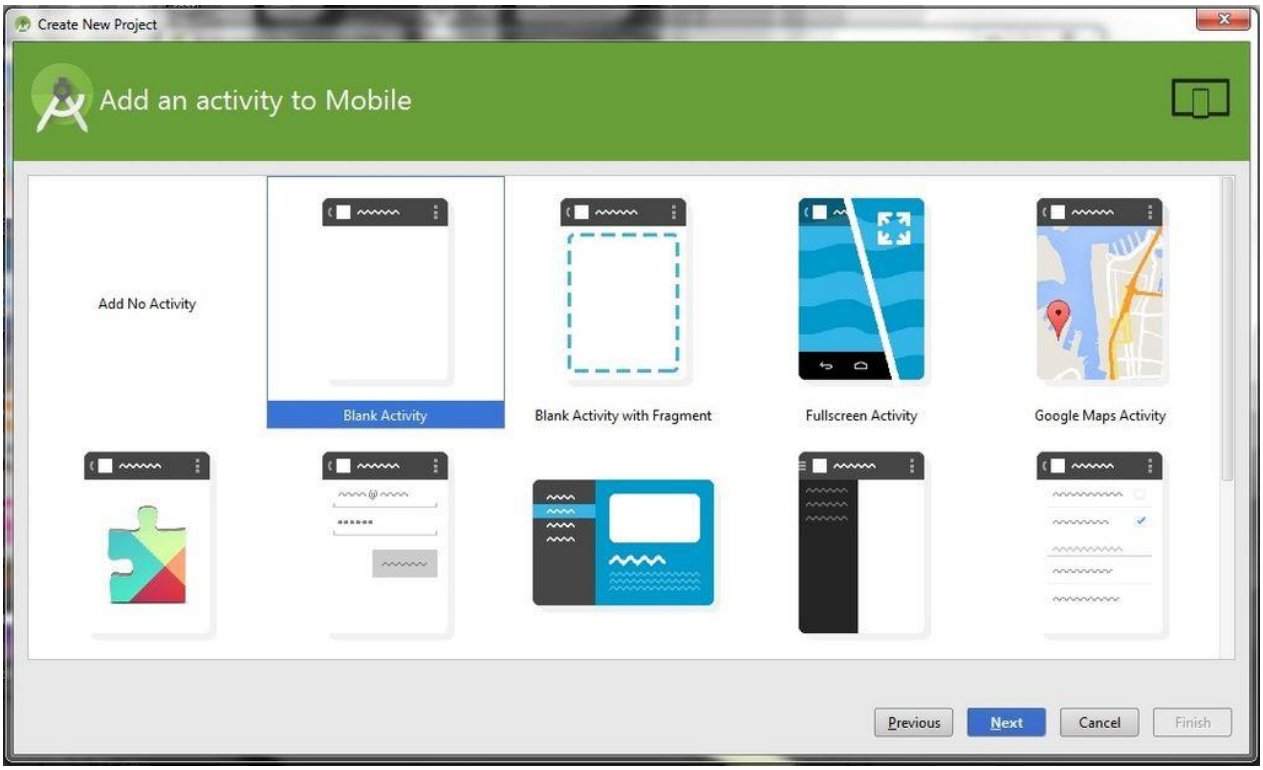
2.2.1 Creating an App Using Android Studio

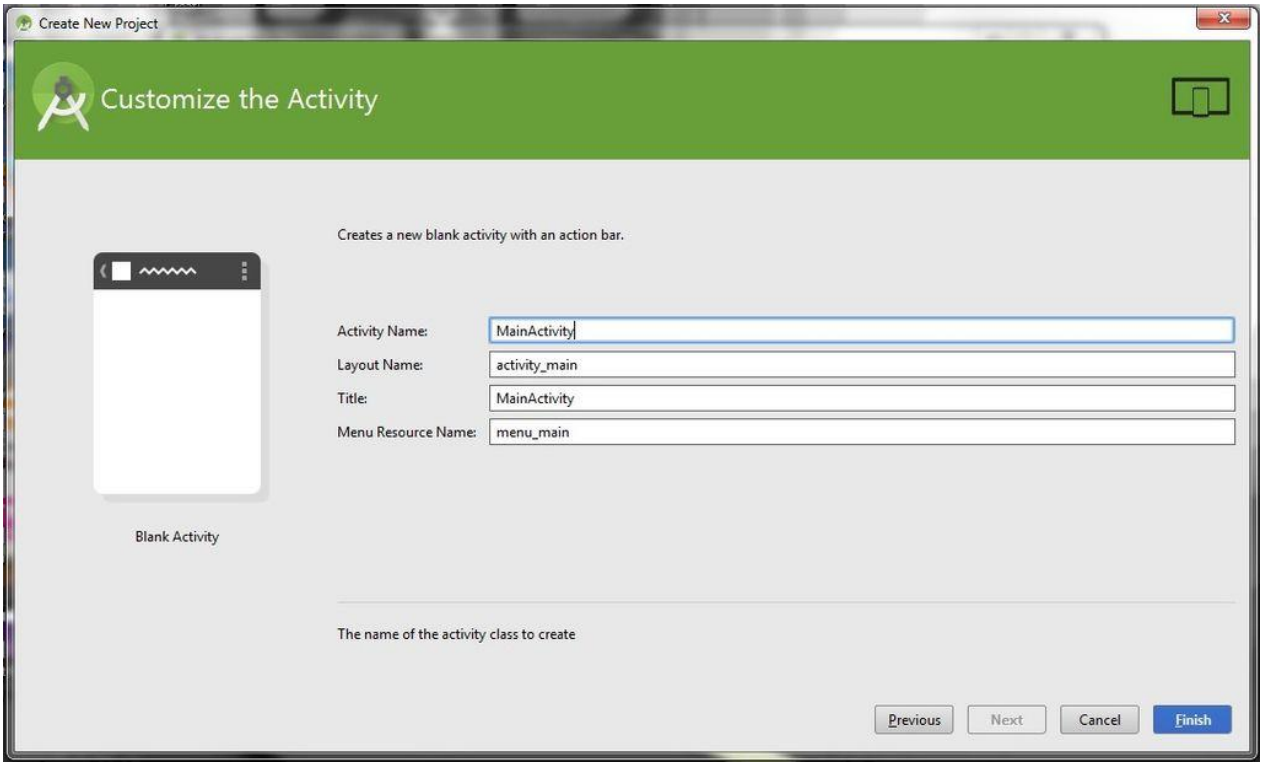
1. Open Android Studio.
2. Under the "Quick Start" menu, select "Start a new Android Studio project."
3. On the "Create New Project" window that opens, name your project "HelloWorld".
4. If you choose to, set the company name as desired*.
5. Note where the project file location is and change it if desired.
6. Click "Next."
7. Make sure on that "Phone and Tablet" is the only box that is checked.
8. If you are planning to test the app on your phone, make sure the minimum SDK is below your phone's operating system level.
9. Click "Next."
10. Select "Blank Activity."
11. Click "Next."
12. Leave all of the Activity name fields as they are.
13. Click "Finish."




Create New Project
New Project
Android Studio
Configure your new project
Application name:
HelloWorld
Company Domain:
thezachbales.example.com
Package name:
com.example.thezachbales.helloworld
Edit
Project location:
C:\Users\TheZachBales\AndroidStudioProjects\HelloWorld
Previous
Next
Cancel
Finish


Create New Project
Target Android Devices
Select the form factors your app will run on
Different platforms require separate SDKs
Phone and Tablet
Minimum SDK
API 15: Android 4.0.3 (IceCreamSandwich)
Lower API levels target more devices, but have fewer features available. By targeting API 15 and later, your app will run on approximately 90.4% of the devices that are active on the Google Play Store. Help me choose.
TV
Minimum SDK
API 21: Android 5.0 (Lollipop)
Wear
Minimum SDK
API 21: Android 5.0 (Lollipop)
Glass (Not Installed)
Minimum SDK
Previous
Next
Cancel
Finish


Create New Project
Add an activity to Mobile
Add No Activity
Blank Activity
Blank Activity with Fragment
Fullscreen Activity
Google Maps Activity
Previous
Next
Cancel
Finish


Create New Project
Customize the Activity
Creates a new blank activity with an action bar.
Activity Name:
MainActivity
Layout Name:
activity_main
Title:
MainActivity
Menu Resource Name:
menu_main
Blank Activity
The name of the activity class to create
Previous
Next
Cancel
Finish

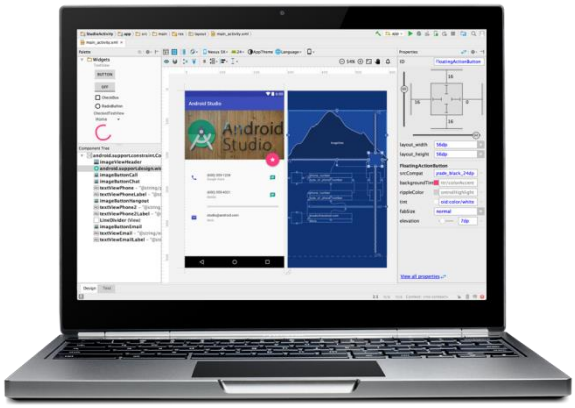
2.3 Xamarin Studio

Xamarin Studio is a modern, sophisticated IDE with many features for creating iOS, Mac and Android applications. It includes a rich editor, debugging, native platform integration with iOS, Mac and Android, and integrated source control to name just of few of its many features.

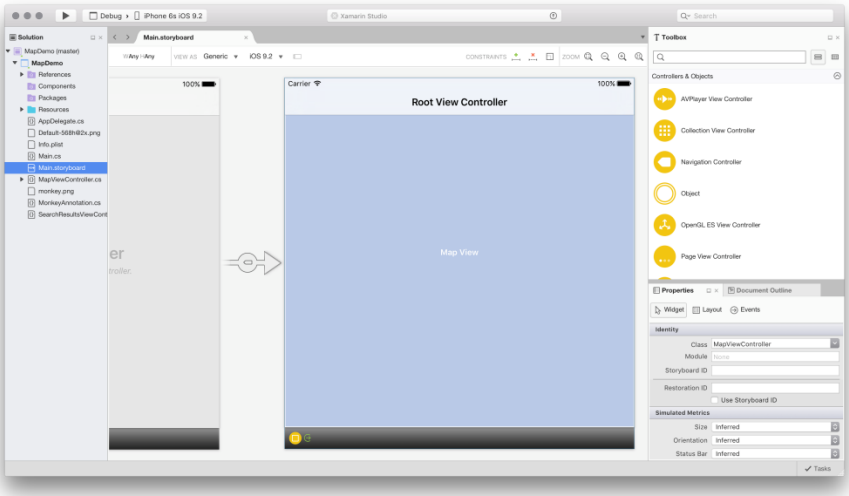
Xamarin, the San Francisco-based company that allows developers to code applications for a variety of different platforms using **Microsoft's** C# programming language, announced today that it has raised a $54 million Series C round of financing.

Conclusion

Xamarin is the best IDE to develop app any mobile device as it support development for iOS, Android and Windows simultaneously with same code written in C#. It overtakes every IDE present currently in the market.



Android Studio



Xamarin Studio

Chapter 3: Android Booting

3.1 Key bootup components are as follows:

3.1.1 Bootloader

The first program which runs on any Android system is the bootloader. Technically, the bootloader is outside the realm of Android itself, and is used to do very low-level system initialization, before loading the Linux kernel. The kernel then does the bulk of hardware, driver and file system initialization, before starting up the user-space programs and applications that make up Android.

Often, the first-stage bootloader will provide support for loading recovery images to the system flash, or performing other recovery, update, or debugging tasks.

The bootloader on the ADP1 detects certain keypresses, which can be used to make it load a 'recovery' image (second instance of the kernel and system), or put the phone into a mode where the developer can perform development tasks ('fastboot' mode), such as re-writing flash images, directly downloading and executing an alternate kernel image, etc.

3.1.2 'init'

A key component of the Android bootup sequence is the program 'init', which is a specialized program for initializing elements of the Android system. Unlike other Linux systems (embedded or otherwise), Android uses its own initialization program. (Linux desktop systems have historically used some combination of /etc/inittab and sysV init levels - e.g. /etc/rc.d/init.d with symlinks in /etc/rc.d/rc.[2345]). Some embedded Linux systems use simplified forms of these -- such as the init program included in busybox, which processes a limited form of /etc/inittab, or a direct invocation of a shell script or small program to do fixed initialization steps.

The Android 'init' program processes two files, executing the commands it finds in them, called 'init.rc' and 'init.<machine_name>.rc', where <machine_name> is the name of the hardware that Android is running on. (Usually, this is a code word. The name of the HTC1 hardware for the ADP1 is 'trout', and the name of the emulator is 'goldfish'.

The 'init.rc' file is intended to provide the generic initialization instructions, while the 'init.<machine_name>.rc' file is intended to provide the machine-specific initialization instructions.

3.1.3 'init' resources

The syntax for these .rc files is documented in a readme file in the source tree. You may be able to reconstruct it though:

- You need a local copy of the AOSP sourcetree, and run the usual *build/envsetup.sh* preparation
- Use *repo init -b* to check out the AOSP sourcetree with a tag around 2.3, then *make sdk sdk_all*
- This worked for me, though I used some tag around 4.0.4. Some links had to be fixed in the resulting html output

3.2 Sequence of boot steps on ADP1

3.2.1 firmware

- first-stage bootloader runs
  - it detects if a special key is held, and can launch the recovery image, or the 'fastboot' bootloader
- eventually, a kernel is loaded into RAM (usually with an initrd)
  - Normally, this will be the kernel from the 'boot' flash partition.

3.2.2 kernel

- the kernel boots
  - core kernel initialization
    - memory and I/O areas are initialized
    - interrupts are started, and the process table is initialized
  - driver initialization
  - kernel daemons (threads) are started
  - root file system is mounted
  - the first user-space process is started
    - usually /init (note that other Linux systems usually start /sbin/init)

3.2.3 user space

- the kernel runs /init
  - /init processes /init.rc and /init.<machine_name>.rc
  - Dalvik VM is started (zygote).
  - several daemons are started:
    - rild - radio interface link daemon
    - vold - volume daemon (media volumes, as in file systems - nothing to do with audio volume)
- the system_server starts, and initializes several core services
  - nitalization is done in 2 steps:
    - 1) a library is loaded to initialize interfaces to native services, then
    - 2) java-based core services are initialized in ServerThread::run() in SystemServer.java
- the activity manager starts core applications (which are themselves dalvik applications)
  - com.android.phone - phone application
  - android.process.acore - home (desktop) and a few core apps.
- other processes are also started by /init, somewhere in there:
  - adb
  - mediaserver
  - dbus-daemon
  - akmd

Chapter 4: Getting Started with Android Application

4.1 Android Application Components

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application:

Components	Description
Activities	They dictate the UI and handle the user interaction to the smart phone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

Activities

An activity represents a single screen with a user interface, in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of **Activity** class as follows –

```
public class MainActivity extends Activity {
}
```

Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of **Service** class as follows −

```
public class MyService extends Service {
}
```

Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and each message is broadcaster as an **Intent** object.

```
public class MyReceiver  extends  BroadcastReceiver {
     public void onReceive(context,intent){}
}
```

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver*class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
}
```

4.2 Android Manifest.xml

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

Among other things, the manifest file does the following:

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

Implementing Manifest.xml file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.yash.helloworld" android:versionCode="1"
        android:versionName="1.0" >
          <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />

                <application android:icon="@drawable/myicon"
                                    android:label="@string/helloworld"
                                   android:theme="@style/AppTheme" >

                      <activity android:name=".MainActivity"
                              android:label="@string/firstpageheading" >
```

```
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
        </application>
    </manifest>
```

Elements of the AndroidManifest.xml file

<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.

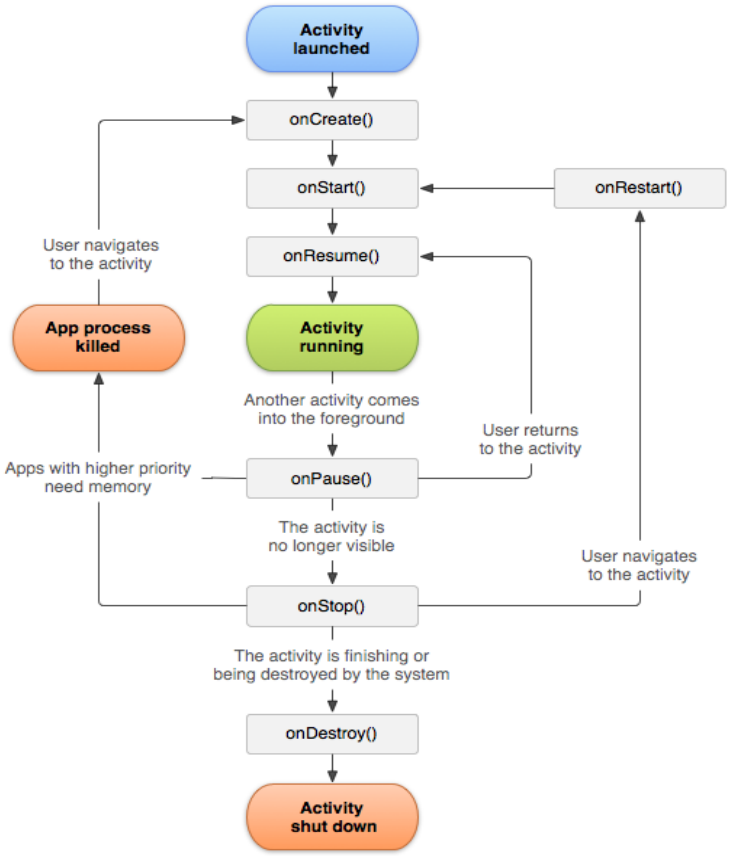
<action>

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.

4.3 Activity Life Cycle



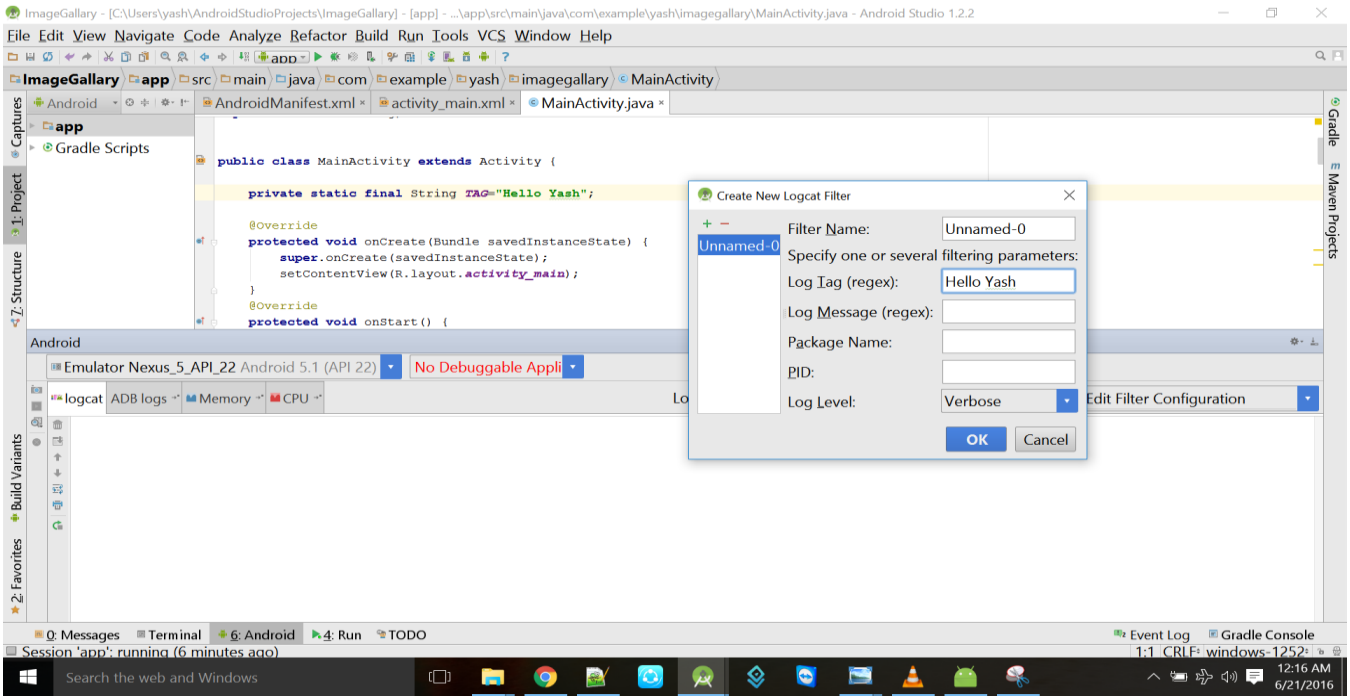
Program to demonstrate Activity LifeCycle

```
package com.example.yash.lifecycleofAndroid;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

        private static final String TAG="Hello Yash ";

   @Override
   protected void onCreate(Bundle savedInstanceState)
        {       super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main); //Layout will be explained later
        }
   @Override
   protected void onStart() {
      super.onStart();
      Log.i(TAG, "onStart()");
   }
```

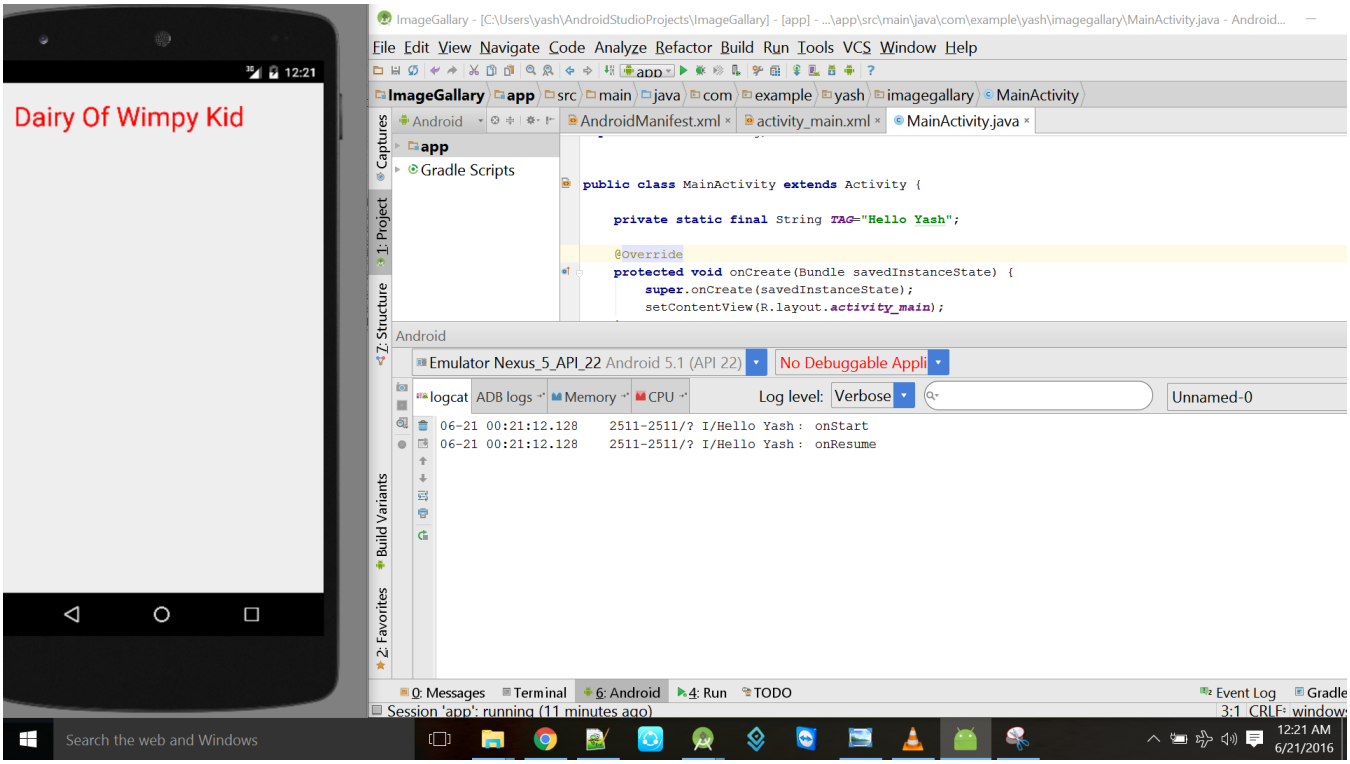
```
    @Override
    protected void onResume() {
        super.onResume();
        Log.i(TAG, "onResume()");
    }


    @Override
    protected void onPause() {
        super.onPause();
        Log.i(TAG, "onPause()");
    }


    @Override
    protected void onStop() {
        super.onStop();
        Log.i(TAG, "onStop()");
    }


    @Override
    protected void onRestart() {
        super.onRestart();
        Log.i(TAG, "onRestart()");
    }


    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy()");
    }


    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        Log.i(TAG, "onSaveInstanceState()");
    }


    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);
        Log.i(TAG, "onRestoreInstanceState()");
    }

}
```

OUTPUTS

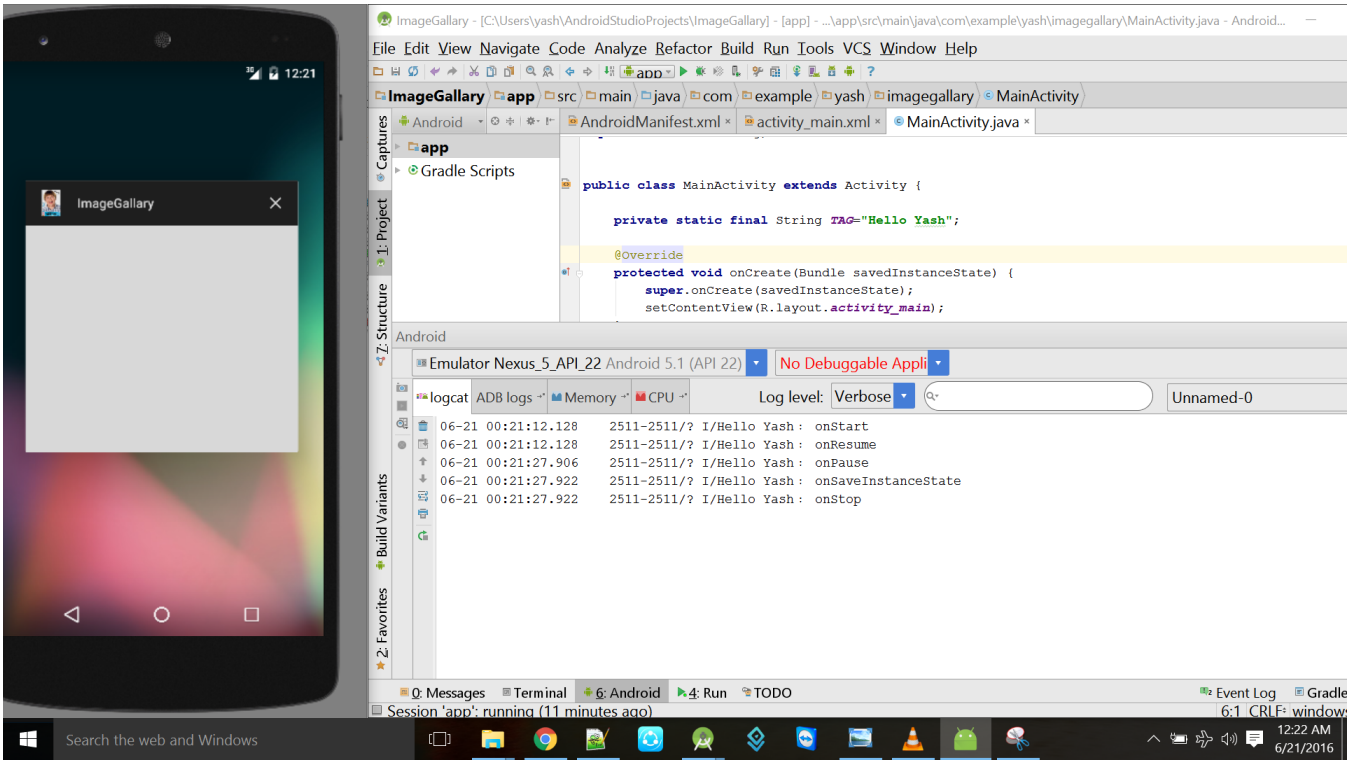
Tracking the “Hello Yash” keyword to debug the Application.



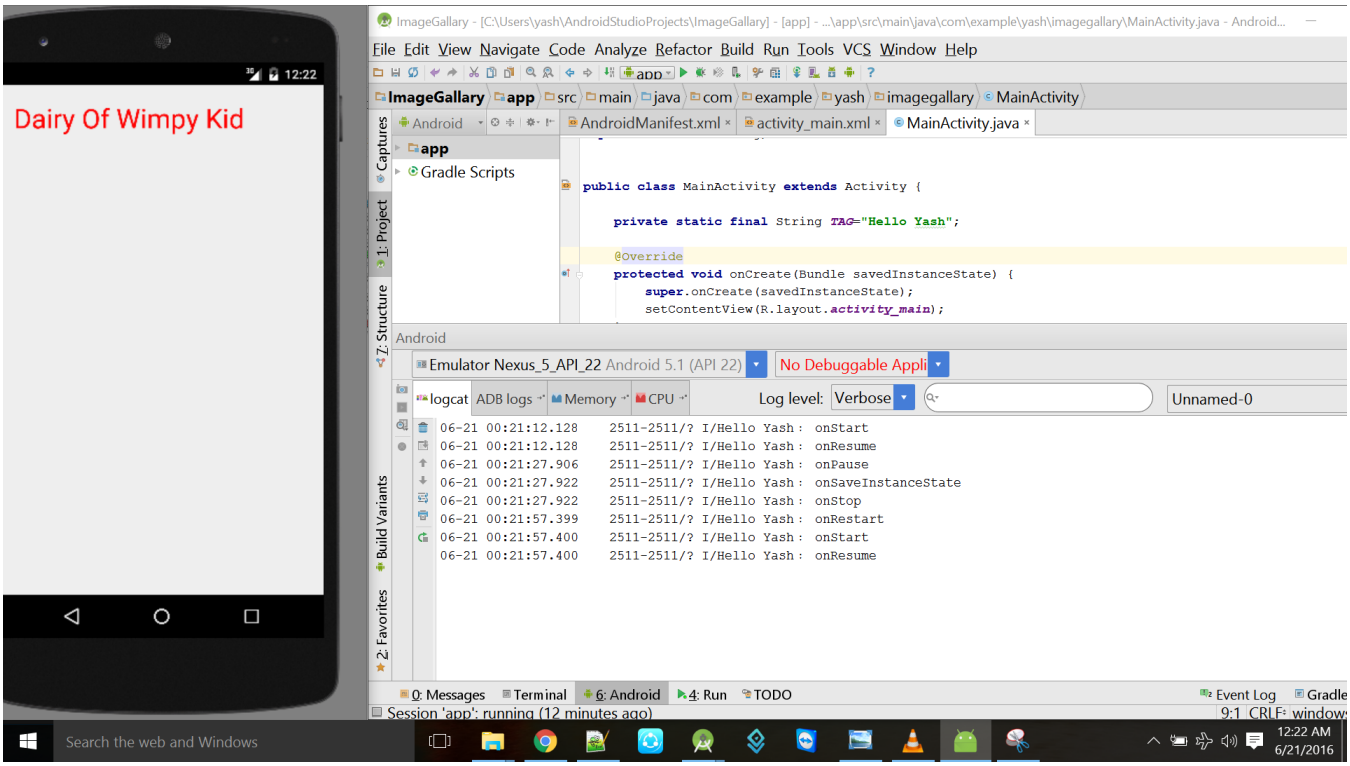
Running the App



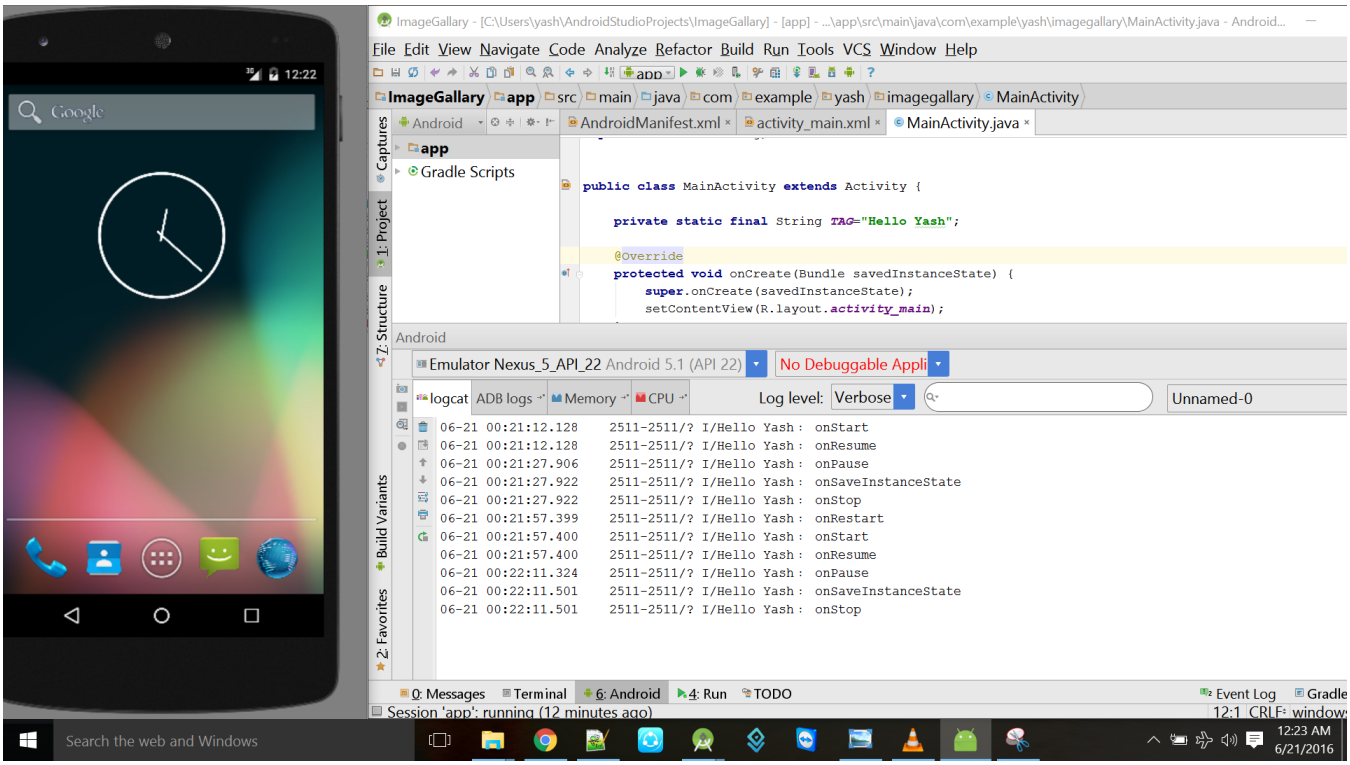
Right button pressed □



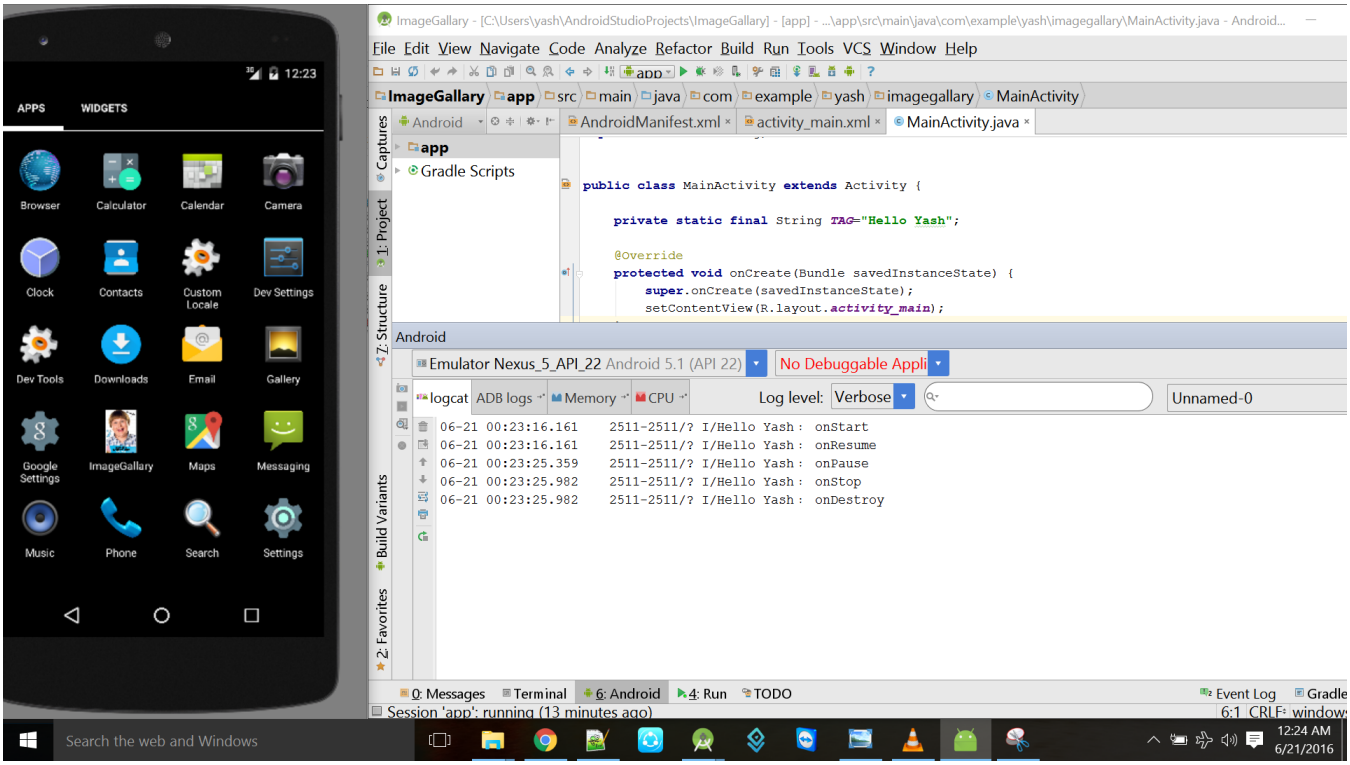
Restarting the App by selecting the screen again



Middle button pressed ○



Restarting the App and than Left/Back button pressed ◄



4.5 First Android Application

There are 3 essential files required to make any Android Application:

- manifest.xml - Linking the Application with the Operating System
- Activity.java - Activity and Process Implementation
- Implementation activity_main.xml - UI design Implementation

Manifest.xml (already explained Earlier)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.yash.helloworld" android:versionCode="1"
        android:versionName="1.0" >

       <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />
              <application android:icon="@drawable/myicon"
                                  android:label="@string/helloworld"
                                 android:theme="@style/AppTheme" >

                     <activity android:name=".MainActivity"
                              android:label="@string/firstpageheading" >

                        <intent-filter>
                     <action android:name="android.intent.action.MAIN" />
                     <category android:name="android.intent.category.LAUNCHER" />
                         </intent-filter>
                     </activity>
              </application>
        </manifest>
```

Activity.java

```
package com.example.yash.firstapp;
import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

activity_main.xml

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

  < TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:padding="@dimen/padding_medium"
    android:text=”Hi there"
    tools:context=".MainActivity" />

</ RelativeLayout>
```

OUTPUT