**KAMARAJ**
**COLLEGE OF ENGINEERING & TECHNOLOGY**
S.P.G. Chidambara Nadar - C. Nagammal Campus, S.P.G.C. Nagar, VIRUDHUNAGAR

Faculty Development Programme
On

# MOBILE APPLICATION DEVELOPMENT

2nd to 4th December, 2015

Don't Deliver a Product. Deliver an Experience

# KAMARAJ
## COLLEGE OF ENGINEERING & TECHNOLOGY
S.P.G. Chidambara Nadar - C. Nagammal Campus, S.P.G.C. Nagar, VIRUDHUNAGAR

**ABOUT THE COLLEGE**

KAMARAJ COLLEGE OF ENGINEERING AND TECHNOLOGY is the outcome of the cherished desire and dream of the people of Virudhunagar. It is rightly named after the son of this soil, Bharat Ratna Sri. K. Kamaraj Who was the undisputed leader of the suppressed and downtrodden and a champion of free education for the masses. Our college was established in the year 1998.

The College is a co-educational institution. It offers 10 UG courses in Engineering (CSE, ECE, EIE, EEE, Civil, Mechanical, Mechatronics, Information Technology, Polymer Technology and Bio Technology), 8 PG courses in engineering (Biotechnology, Polymer Science and Technology, Communication and Networking, Manufacturing Engineering, Structural Engineering, Power Systems Engineering, Computer Science and Engineering and Biometrics & Cyber Security) a 5 year integrated M.Sc. (Software Systems) and Ph.D. programme through the 5 Research Centers(Biotechnology, Polymer Technology, Electrical and Electronics Engineering, Chemistry and Physics).

**Prepared by**

Er. A. John Paul Antony
Assistant Professor / CSE
Apple Lab Co-ordinator
Kamaraj College of Engineering and Technology
Virudhunagar - 626001.

# Table of Contents

## 12. RSS Feed

12.1 Rss Feed
12.2 Running the Rss Feed App

## 13. Testing and Publishing

13.1 Debugging
13.2 Android App Development Life Cycle
13.3 APK Conversion Process
13.4 App Publishing Guidance
13.5 Tips for Launching an App

## 14. Intel XDK

14.1 Getting started with Intel XDK
14.2 Edit Your App
14.3 Testing and Debugging
14.4 Package Options
14.5 Build Your App

# 1. Introduction to Android

## Objectives

After completing this chapter we will be able to understand the following topics.

- Android Introduction
- Development Environment
- Creating an App in Android Studio
- Anatomy of Android Project
- Running the First App
- Ingredients of an Android App

## 1.1 Android Introduction

Android is a mobile operating system that is based on a modified version of Linux. Android, as a system, is a Java-based operating system that runs on the Linux kernel. The system is very lightweight and full featured.

It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work. Google wanted Android to be open and free; hence, most of the Android code was released under the open source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android.

**Android Versions**

Android has gone through quite a number of updates since its first release. The follow table shows the various versions of Android and their codenames.

| ANDROID VERSION | RELEASE DATE | CODENAME |
|---|---|---|
| 1.1 | 9 February 2009 | |
| 1.5 | 30 April 2009 | Cupcake |
| 1.6 | 15 September 2009 | Donut |
| 2.0/2.1 | 26 October 2009 | Eclair |
| 2.2 | 20 May 2010 | Froyo |
| 2.3 | 6 December 2010 | Gingerbread |
| 3.0/3.1/3.2 | 22 February 2011 | Honeycomb |
| 4.0 | 19 October 2011 | Ice Cream Sandwich |

**Features of Android**

As Android is open source and can be customized by the manufacturers, there are no fixed hardware or software configurations. Android itself supports the following features:

- **Storage** — Uses SQLite, a lightweight relational database, for data storage.
- **Connectivity** — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi- Fi, LTE, and WiMAX. **Messaging** — Supports both SMS and MMS.
- **Web browser** — Based on the open source WebKit, together with Chrome's V8 JavaScript engine.
- **Media support** — Includes support for the following media: H.263, H.264 (in 3GP or MP4
- container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or
- 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP
- **Hardware support** — Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
- **Multi-touch** — Supports multi-touch screens
- **Multi-tasking** — Supports multi-tasking applications
- **Flash support** — Android 2.3 supports Flash 10.1.
- **Tethering** — Supports sharing of Internet connections as a wired/wireless hotspot

**Android Architecture**

**Linux Kernel**

Since the android architecture is built on top of Linux Kernel, the Kernel is used as a hardware abstraction layer. i.e, It can be used as an abstraction layer between hardware and software. For example, consider the battery function. The devices which run on android will have hardware component like camera, blue tooth, battery etc. The power management interface from the Kernel will interact with the battery (hardware). There is a list of low level interfaces provided by the Kernel such as camera driver, audio drivers, memory management, display driver etc.

**Libraries**

The libraries are the APIs which contains the features of android operating system.

Android Runtime Contains two elements

- Dalvik Virtual Machine
- Core Libraries

**Dalvik Virtual Machine**

The Dalvik Virtual Machine (DVM) is available under Android runtime. The Dalvik virtual machine executes .dex files. The .dex files are provided by DX tool which works as a compiler for android SDK. This DX tool is explained in android SDK tools. The Dalvik virtual machine (DVM) does the job of Java virtual machine (JVM).

The role of Dalvik virtual machine is similar to java virtual machine. While java virtual machine executes byte code, on the other side, in android programming Dalvik virtual machine executes Dalvik byte code (.dex files).

**Core Libraries**

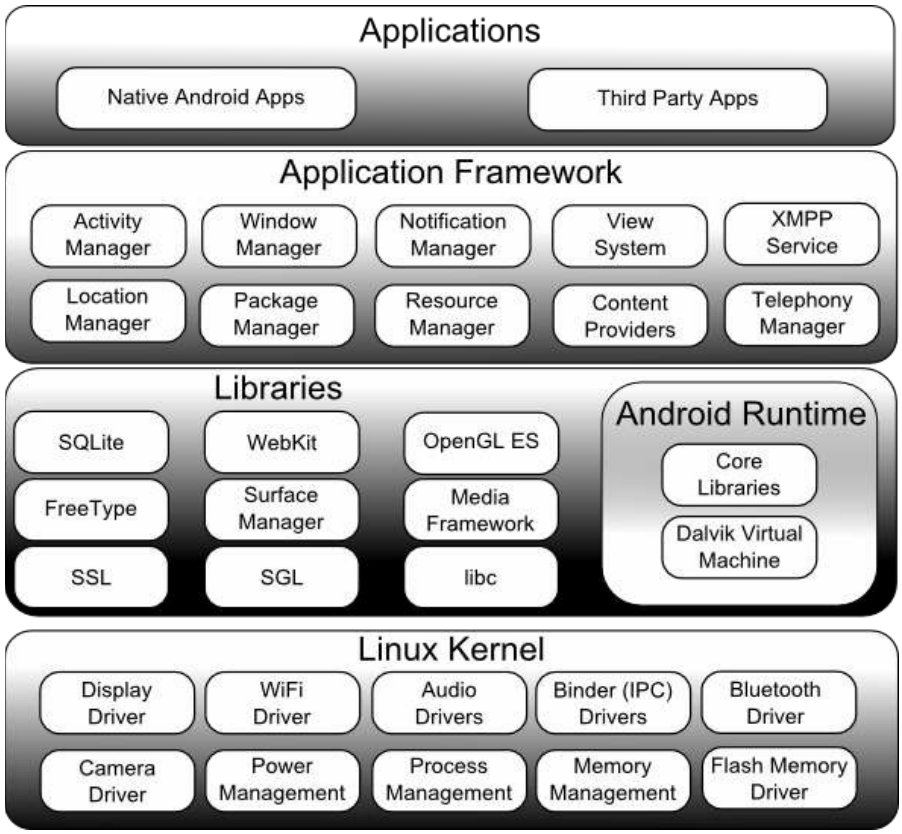
Core libraries are core API s of java programming language which provides powerful interfaces for implementing file access, network access, utility, data structures etc.

**Application Framework**

The application framework contains libraries for various areas like power management, multimedia, storage access and graphical user interface. The libraries in the framework are written in Java and run on top of the core libraries of the Android Runtime. The application framework of android provides managers for various purposes such as resources handling, content providers, system wide notification etc. The android applications are using the services of these managers.

**Application**

On Android, every application runs in its own process, each of which runs in its own virtual machine (VM). Applications are the android applications which are executed on a device. The applications are the top layer in the android architecture. This layer will consists of pre-installed and our own applications. The android application we develop goes into this layer.




## 2.2 Development Environment

The android development environment consists of the following.

- Android Software Development Kit - Android SDK
- Android Development Tools - ADT
- Integrated Development Environment - IDE

## 1. Android Software Development Kit (SDK)

The Android SDK includes a variety of tools that help us develop mobile applications for the Android platform. The tools are classified into two groups:

a) SDK tools
b) Platform tools

SDK tools are platform independent and are required no matter which Android platform we are developing on. Platform tools are customized to support the features of the latest Android platform.

a) **SDK Tools**

SDK tolls are periodically updated. SDK tools include the

- Android SDK Manager (android SDK)
- AVD Manager (android AVD)
- Emulator (emulator)
- Dalvik Debug Monitor Server (DDMS)
- Other frequently-used SDK tools

**Android SDK Manager**

Android SDK Manager is provided for the purpose of adding required packages or android APIs to the Base IDE. i.e. To download and install APIs we can use android SDK manager. It separates APIs, platform - tools, development tools.

In order to compile our application against a particular version of Android, we must use the SDK Manager to download and install the SDK Platform for that release. If we want to test our application on an emulator, we must also download at least one System Image for that Android version.

**AVD Manager**

The AVD Manager stands for Android Virtual Device Manager. As the name implies, the AVD Manager is used for creating and managing Android Virtual Devices.

**Emulator**

This is one of the most important tools provided by android SDK. The emulator is a virtual mobile device that runs on the computer. It can be used to test our android applications. It allows the developer to test and run the android applications without using a physical mobile device during development stage. The emulator for each android project can be configured with the help of Android Virtual Device (AVD) Manager.

**Dalvik Debug Monitor Server - DDMS**

Dalvik Debug Monitor Server (DDMS) is a debugging tool. It provides port-forwarding services, screen capture on the device, thread and heap information on the device, incoming call and SMS spoofing, location data spoofing, logcat, process, and radio state information, etc. DDMS works with both the emulator and a connected device. If both are connected and running simultaneously, DDMS defaults to the emulator. Let us discuss about DDMS in detail when exploring the IDE.

**Other frequently used SDK Tools**

**MKSDCARD**

MKSDCARD is a useful tool and available as a .exe file. Assume we are developing an application to read or write file from or to SD memory card in a mobile device. To test that application in an emulator this tool is invoked. i.e it create a small partition on our hard drive (the system we use to develop applications) to hold and retain the files during testing of an application. When we test our application using emulator, the emulator will consider the small partition as a SD memory card.

**Systrace**

The Systrace tool helps analyze the performance of our application by capturing and displaying execution times of our applications processes and other Android system processes. The tool combines data from the Android kernel such as the CPU scheduler, disk activity, and application threads to generate an HTML report that shows an overall picture of an Android device's system processes for a given period of time. The Systrace tool is particularly useful in diagnosing display problems where an application is slow to draw or stutters while displaying motion or animation.

**Device Monitor**

Android Device Monitor is a stand-alone tool that provides a graphical user interface for several Android application debugging and analysis tools. It encapsulates the following tools:

- DDMS
- Tracer for OpenGL ES
- Hierarchy Viewer
- Systrace
- Traceview

b) **Platform Tools**

The platform tools are generally updated each time we install a new SDK platform. Each update of the platform tools is backward compatible with older platforms. Usually, we directly use only one of the platform tools—the Android Debug Bridge (adb). The other platform tools, such as aidl, aapt, dexdump, and dx, are typically called by the Android build tools or Android Development Tools (ADT), so we rarely need to invoke these tools directly.

**Android Debug Bridge (ADB)**

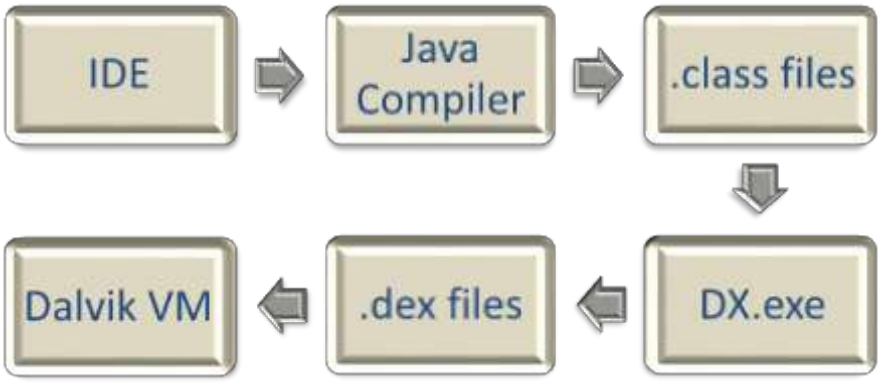
Android Debug Bridge is a versatile tool that lets us manage the state of an emulator instance or Android-powered device. We can also use it to install an Android application (.apk) file on a device.

Android Debug Bridge controls both emulators and devices. It allows us to install and remove programs in emulators and android devices. Simply it acts as bridge between the system and the android device to establish communication between them. Through this communication we can get connected to the android device and we can perform the actions such as moving and synchronising the files to the device, reading the device memory usage etc. Actually it is a command line tool. The complex commands can be executed through command line mode using the commands available for ADB.

**DX.exe**

The DX.exe file is one of the most useful tools. These .dex files are understood by android devices. The DX.exe works as compiler of the android SDK. It creates .dex extensions files when we are running the java files. The .java files written in IDE, are compiled with the help of java compiler. The byte code files are read and converted as .dex (dalvik executable) format by DX tool.

While writing java programs, the code is compiled into byte code and jvm executes that byte code. On the other side, in android programming we still write java code and it is compiled to a byte code format. Finally the byte code is converted to Dalvik executable format (.dex). This .dex files are read by the Dalvik virtual machine.




## 2. Android Development Tools (ADT)

Android Development Tools (ADT) is a plug-in for the Eclipse IDE that provides a suite of tools that are integrated with the Eclipse IDE. It offers access to many features that are helpful for developing Android applications. It extends the capabilities of Eclipse to let us quickly set up new Android projects, create an application UI, add packages based on the Android Framework API, debug our applications using the Android SDK tools, and even export signed (or unsigned) .apk files in order to distribute our application. Before we install or use ADT, we need to check that whether we have compatible versions of both the Eclipse IDE and the Android SDK installed.

ADT provides GUI access to many of the command line SDK tools as well as a UI design tool for rapid prototyping, designing, and building our application's user interface.

### 3. **IDE - Android studio**

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA.

It has the android build system. The Android build system is the toolkit used to build, test, run and package the apps. This build system replaces the Ant system used with Eclipse ADT. It can run as an integrated tool from the Android Studio menu and independently from the command line.

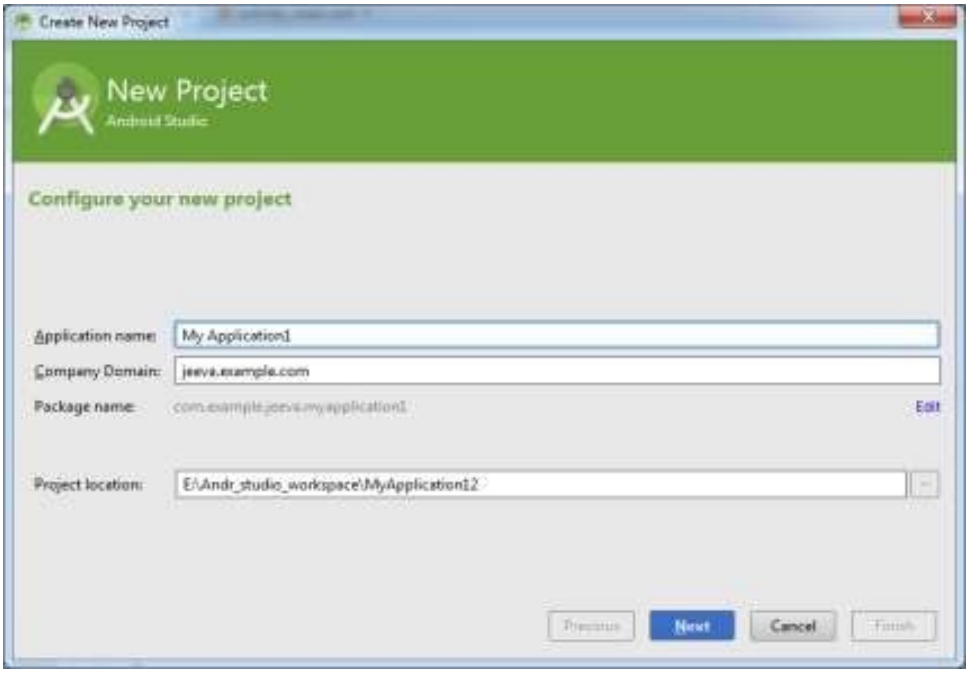
Android Studio provides a memory monitor view so the app's memory usage such as to find deallocated objects, locate memory leaks and track the amount of memory the connected device is using can be monitored. Some of the features of android studio are given below.

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help in building common app features
- Rich layout editor with support for drag and drop theme editing

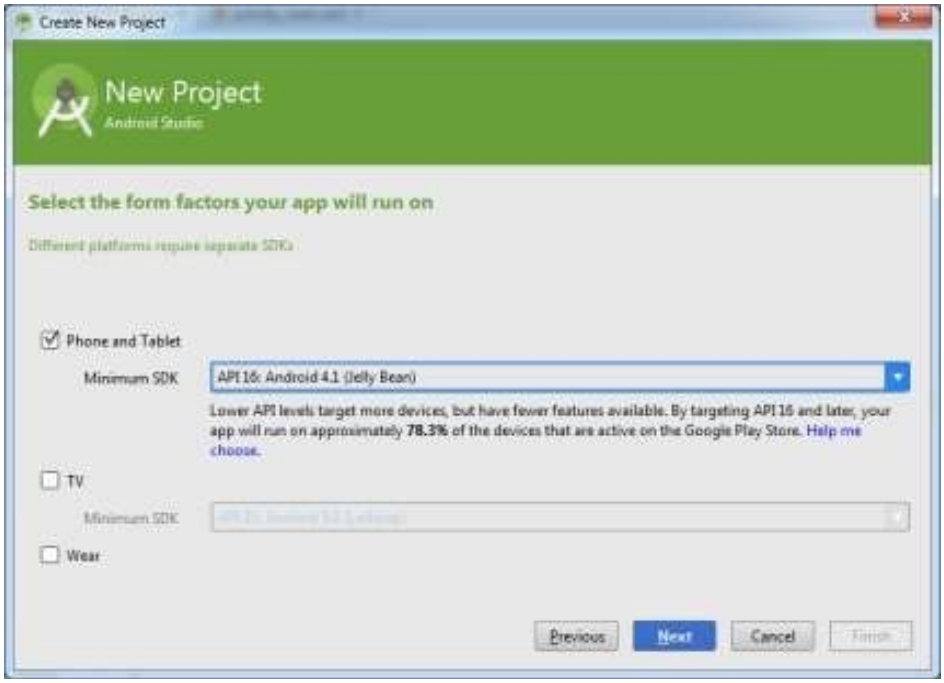
## 1.3. Creating an App in Android Studio

**Creating a project**

To create a new project in Android Studio select the File menu and choose 'New Project' option. (File -> New project). Then a dialog box will appear as follows.



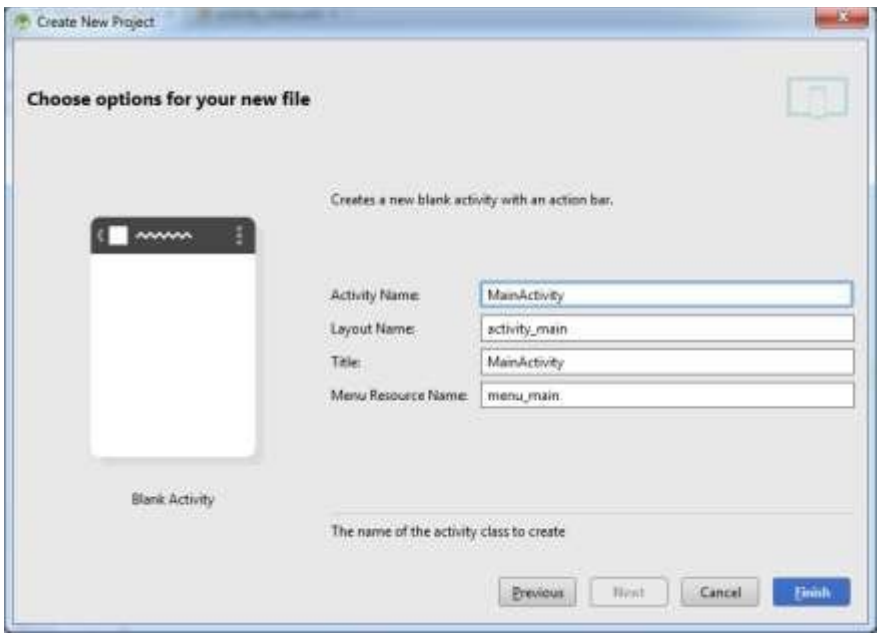

In the dialog box, it prompts for the project name which is indicated as application name.



The package name is the name of the package where our application code resides. Android Studio provides default package name. If we wish to change the default package name, we can use the edit option given along the package name. The project location is the Android Studio workspace where all our android projects are getting stored. We can change the project location if we desire to choose different location.

After filling the fields, click next to proceed. The next option is choosing the app type. i.e the target device type such as phone or tablets or tv etc. Also choose the minimum API level which requires running our app. After choosing the minimum SDK version click next to proceed.

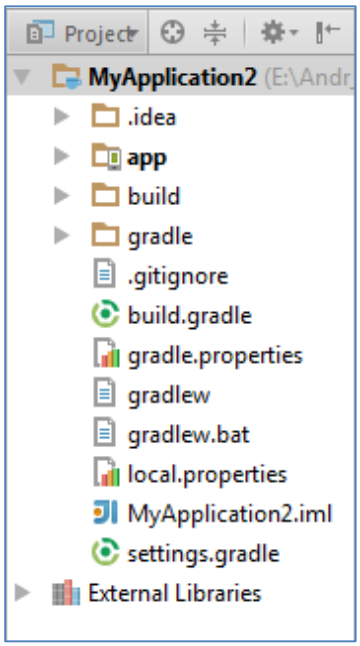

The next option is choosing an activity type. Since each app at least would have one activity, Android Studio provides options to choose an activity type while creating project. We can choose any type of activity we want from the given activity types. In this demo, we will choose 'Blank Activity' type. To proceed with next step click next option. Then a dialog box will appear as shown in the below picture.



The last step in creating project is providing names for activity, layout and menu. The activity name we provide is the name of the activity file which is created under src directory.

## 1.4 Anatomy of Android Project

After we create new project, it is shown in the project explorer. The project folder has the following sub directories



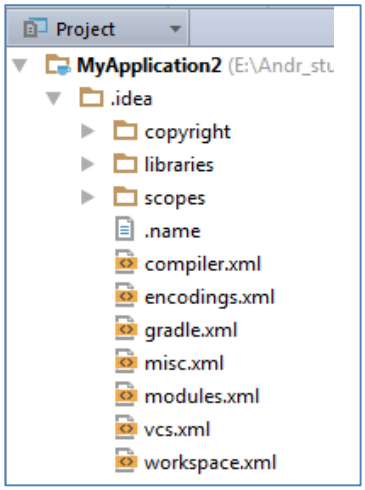

1. .idea
2. app
3. build
4. gradle
5. External Libraries
6. Other files ( .gitignore, build.gradle ,gradle.properties, gradlew, gradlew.bat, local.properties, MyApplication2.iml, settings. gradle)

Main Project

This would be entire project context. A project is an organizational unit that represents a complete software solution. In the above screenshot "MyApplication2" is the name of the project we have created.

Note:

This means that, in theory it is possible to build multiple apps within the same project. Creating multiple apps within the same project doesn't work well. So, it is recommended that not making our hands dirty trying the same thing. Instead, it is a better idea to create single app per single project.



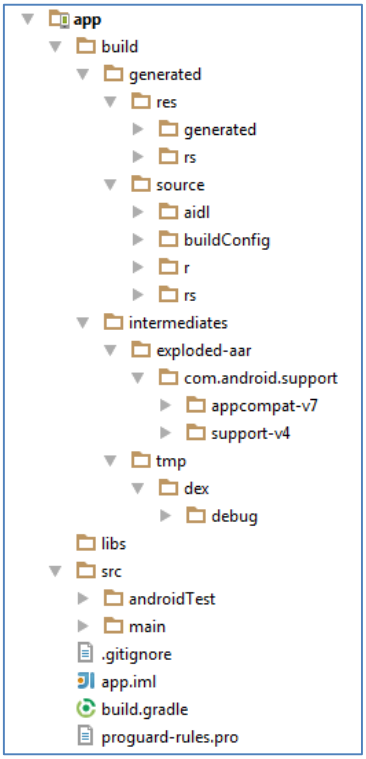

1. **.idea**

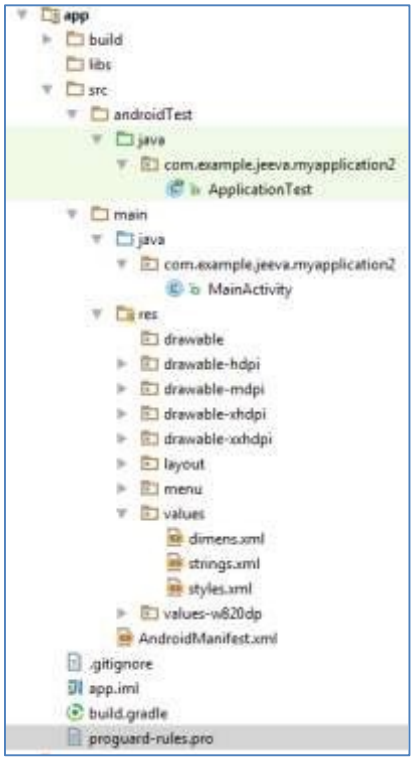
It is the folder for IntelliJ IDEA settings. The .idea folder is created for storing project specific metadata. This means the project specific metadata is stored by Android Studio. In Eclipse the project.properties file does the same thing. The .idea folder contains sub folders copyright, libraries, scopes and files such as .name and xml files.

2. **app**

This is the actual project folder where our application code resides. The application folder has following sub directories.

a) build: This has all the complete output of the make process i.e. classes.dex, compiled classes and resources, etc. In the Android Studio GUI, only a few folders are shown. The important part is that the R.java is found under build/source/r/<build variant>/<package>/R.java




b) libs : This folder holds the libraries or .jar files. It contains private libraries.

c) src: The src folder can have both application code and android unit test script.

The contents of app/src folder

- app/src/androidTest - Contains the instrumentation tests.
- app/src/main/ - Contains Java code source for the app activities, application resources, such as drawable files, layout files, and string values, and manifest file.

app/src/androidTest

The test projects are now automatically part of the app source folder. When a new application module is created, Android Studio creates the src/androidTest . This contains tests for the default configuration and is combined with the debug build type to generate a test application.

These test projects contain Android applications that we write using the Testing and Instrumentation framework. The framework is an extension of the JUnit test framework and adds access to Android system objects.

app/src/main

The main/ folder contain the following list of directories and files.

- java/
- res/
- AndroidManifest.xml

The app/src/main/java folder contains the java code for the application. Our app activities code resides in this folder.

The app/src/main/res folder contains resources for our application. We should always externalize application resources such as images and strings from our code, so that we can maintain them independently. At runtime, Android uses the appropriate resource based on the current configuration. For example, if we need to provide a different UI layout depending on the screen size or different strings depending on the language setting we have to provide them in the resource file. The app/src/ main/res folder contains the following sub directories.

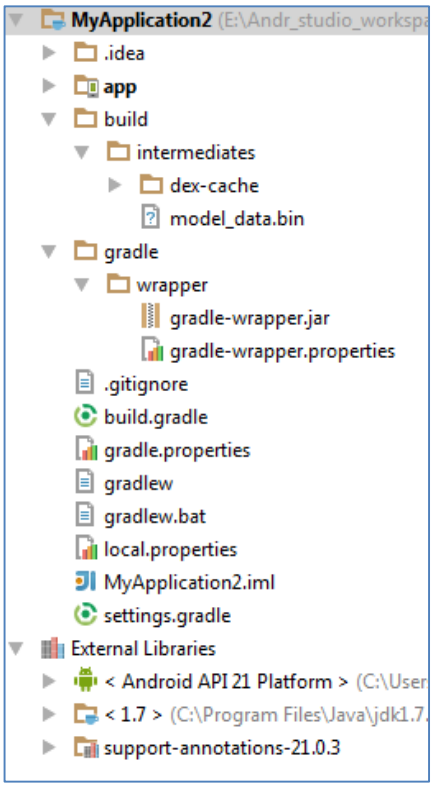
- drawables
- layout
- menu
- values
- AndroidManifest

- drawables - For bitmap files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states.
- layout - Contains XML files that are compiled into screen layouts (or part of a screen). The layout resource defines the architecture for the UI in an Activity or a component of a UI.
- menu - Contains XML files that define application menus. The menu resource defines an application menu such as options menu, context menu, sub menu etc.
- values - For XML files that define resources by XML element type. Unlike other resources in the res/ directory, resources written to XML files in this folder are not referenced by the file name. Instead, the XML element type controls how the resources defined within the XML files are placed into the R class. The values directory contains the following three XML files.

    - dimens.xml
    - strings.xml
    - styles.xml

- AndroidManifest.xml - The control file that describes the nature of the application and each of its components. For instance, it describes: certain qualities about the activities, services, intent receivers, and content providers; what permissions are requested; what external libraries are needed; what device features are required, what API Levels are supported or required; and others. See the AndroidManifest.xml documentation for more information.

.gitignore/ - Specifies the untracked files ignored by git.app.iml/ - IntelliJ IDEA module build.gradle - Customizable properties for the build system. You can edit this file to override default build settings used by the manifest file and also set the location of our key store and key alias so that the build tools can sign our application when building in release mode. This file is integral to the project, so maintain it in a source revision control system.




proguard-rules.pro - ProGuard settings file.

3. **build**
The build folder stores the build output for all project modules.

4. **gradle**
The gradle folder contains gradle wrapper files. This is where the gradle build system's jar wrapper i.e. this jar is how Android Studio communicates with gradle installed in Windows (the OS).

5. **External Libraries**
This is not actually a folder but a place where Referenced Libraries and information on targeted platform SDK is shown.

6. **Other files**
.gitignore – Specifies the files that should be ignored by Git.

- **build.gradle -** This file contains properties for the build system. We can edit this file to specify the default build settings used by the application modules and also set the location of our keystore and key alias so that the build tools can sign our application when building in release mode. This file is integral to the project, so it must be maintained in a source revision control system.
- **gradle.properties -** The gradle.properties file contains project-wide Gradle settings.
- **Gradlew -** The gradlew file is the gradle startup script for Unix.
- **gradlew.bat -** The gradlew.bat file is the gradle startup script for Windows.
- **local.properties -** This file has the properties for the build system, such as the SDK installation path. Since the content of the file is specific to the local installation of the SDK, it should not be maintained in a source revision control system.
- **.iml -** The file with .ml extension stores the module information.
- **settings.gradle -** It specifies the sub-projects to build.

## 1.5 Running the First App

**Setting Emulator**

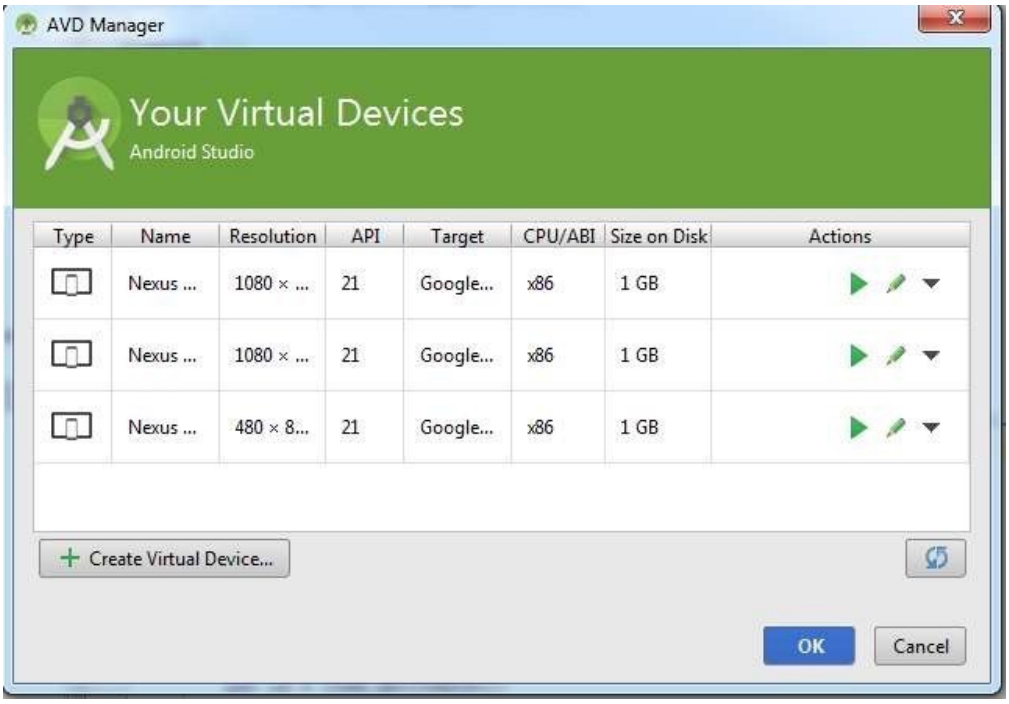
To create an emulator, we need the help of AVD Manager. Since emulator is a virtual device, the AVD Manager (Android Virtual Device Manager) is required to create and manage the emulators.

Android Studio provides tool bar option to open AVD Manager. Click the Android Virtual Device Manager in the toolbar to open it and create new virtual devices for running our app in the emulator.

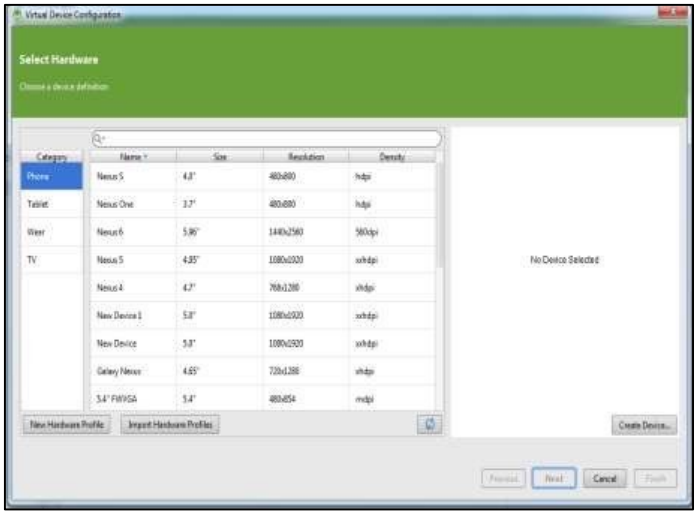
**AVD Manager**

AVD Manager has updated screens with links to help us select the most popular device configurations, screen sizes and resolutions for our app previews. The AVD Manager comes with emulators for Nexus 6 and Nexus 9 devices and also supports creating custom Android device skins based on specific emulator properties and assigning those skins to hardware profiles. Android Studio installs the Intel® x 86 Hardware Accelerated Execution Manager (HAXM) emulator accelerators and creates a default emulator for quick app prototyping.

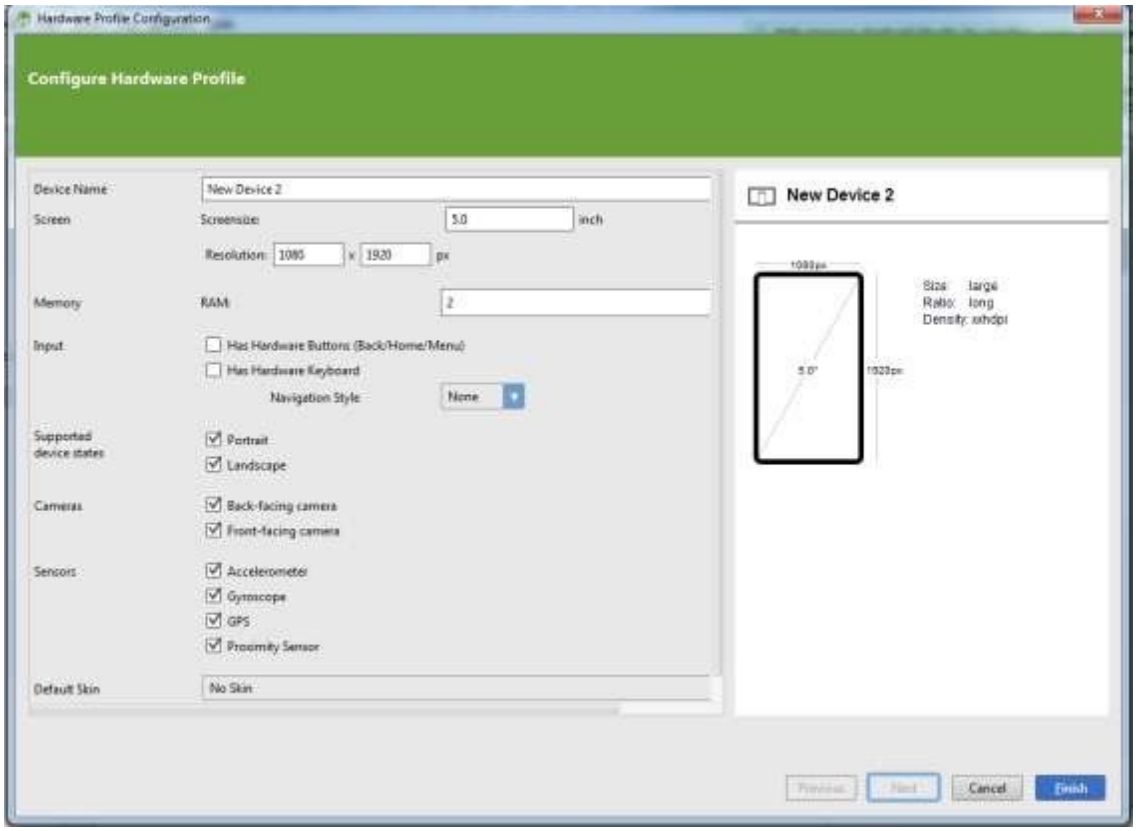
When the AVD Manager icon is selected, the following screen will be shown. This screen will display list of created virtual devices or emulators with the option to create a new one.




To create a new emulator, click on the 'Create Virtual Device' option. The following dialog box will appear with the options to create a new one.

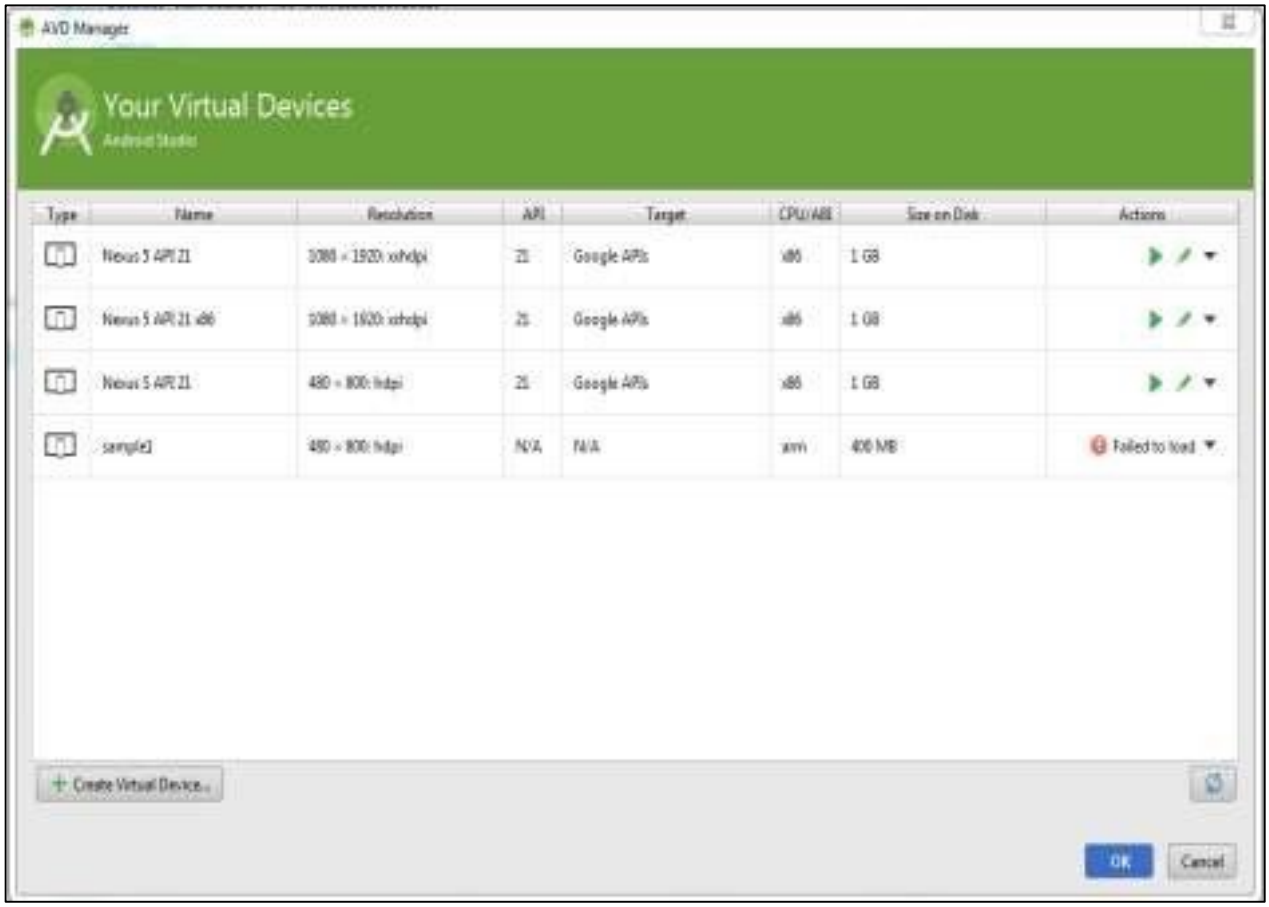


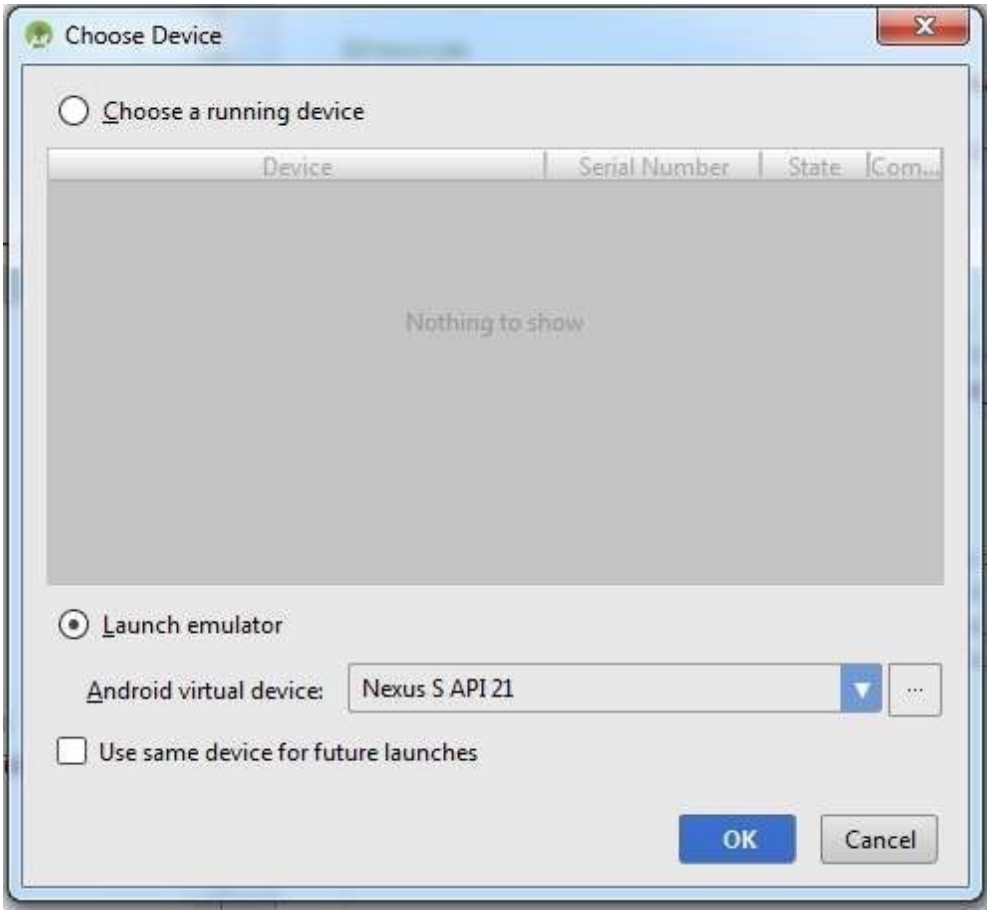
The AVD shows the list of defined hardware profiles such as memory, screen size, resolution etc. To define our own hardware profile, AVD Manager provides an option called ‘New Hardware Profile’. Also we can import the hardware profiles. The following screen shows the option to create new hardware profiles.




We have created an android application. To run that application we have created a virtual device. Now select the run option to run the application. When choosing the run option, a dialog box will appear with the option to launch an emulator or choose the existing (already launched or running) emulator. The AVD Manager allows us to create any number of emulators with the hardware profiles we want. The advantage is, while running an android app we can choose the emulator to run. Emulators are not depending or specific to an android project or application.

The following screen shows the list of created emulators. If an emulator is created successfully, then a run symbol will be shown along with edit and delete options. If the creation of emulator is not successful, then finally the status of emulator will be displayed with error message.




In the above picture we are going to choose the ‘Launch emulator’ option.



The emulator appears with the typical mobile screen. We can watch that our app has been installed. Now if we click on the app icon, it will be launched and the outcome is shown in the above picture. The output is simply showing the text 'Hello World'.

## 1.6 Ingredients of an Android App

The following are the android application components

- Activity
- Intent
- Services
- Broadcast Receivers
- Content Providers

**Activity**

Activity is an individual user interface where visual components can be placed. These visual components are known as Views (also known as widgets). The user can perform various actions by interacting with it. In an android application, the whole window gives the user an interface to interact with and therefore this complete screen makes an Activity. The controls placed in the window allow the user to perform certain actions and are called Views or Widgets.

An application can have more than one activity and each activity operates independently, but can be linked to one another. Each activity must be defined in application's manifest file. Each Activity in android will be subclass of Activity class defined in Android SDK.

**Intent**

Although intents are known as one among the components, actually it is not. It is used to activate a component in an application. For example, in an android application when we want to open a image or video, an option will be shown to the user to choose. The option is for mentioning how to complete the opening action by using which editor the user wants to open.

Here opening of an image or a video is an activity. We want to start another activity called choose the option. To invoke a new activity from our current activity, we need to fire an intent specifying the new activity. And if we want to start other application from our activity, then also we need to fire intent. That is by firing intent, we are telling the android system to make something happen.

**Service**

A service is an android application component that runs in background and has no visual UI. Services are used to perform the processing parts of our application in the background. While the user is working on the foreground UI, services can be used to handle the processes that need to be done in the background. A service can be started by another android application components such as an activity or other services and it will continue to run in the background even after the user switches to another application. Thus services are less likely to be destroyed by Android system to free resources, than Activities.

**Broadcast Receiver**

Broadcast receivers are one of android application components that is used to receive messages that are broadcasted by the android system or other android applications. There are many broadcasts that are initiated by the android system itself and other applications can receive by using Broadcast receiver. Examples of broadcasts initiated by the system are:

- Warning that the battery is getting low
- Screen turned off
- Change of time zone
- The camera has been used to take a picture

**Content Provider**

Content providers in android provide a flexible way to make data available across applications. For example, consider we are creating a to do list in our application, and we are storing it at any storage location such as the data base, file system or in any online storage space. Now through content providers other applications are able to access or even modify the data we have created. In a similar way we can access the data that other utilities have created, by using content providers. Example for content provider in android is the contacts database. The content provider of contacts database allows other applications to query, read, modify and write the contacts info. Android comes with several other built in Content providers that we can use in our application. All content providers are implemented as a subclass of ContentProvider class.

# 2 GUI Components, Fonts and Colours

### Objectives

After completing this chapter we will be able to understand the following topics.

- Activity
- Life Cycle of an Activity
- Creating an Activity
- Intents
- Types of Intents
- GUI Components
- Fonts and Colours

## 2.1 Activity

Activity is the basic building block of any android application. Android applications contain one or more Activities. The introduction of activity has been given in the previous chapter 'Ingredients of an android application'. As we have seen earlier, android activity always has a user interface.

When user launches an application, a window will be displayed. The whole window provides the user interface to the user, this screen makes an activity. . The controls placed in the UI allow the user to do a certain action. The controls in an activity can be created in two different ways. They can be created either by java code or by adding XML code to define the UI. The latter method is always preferred.

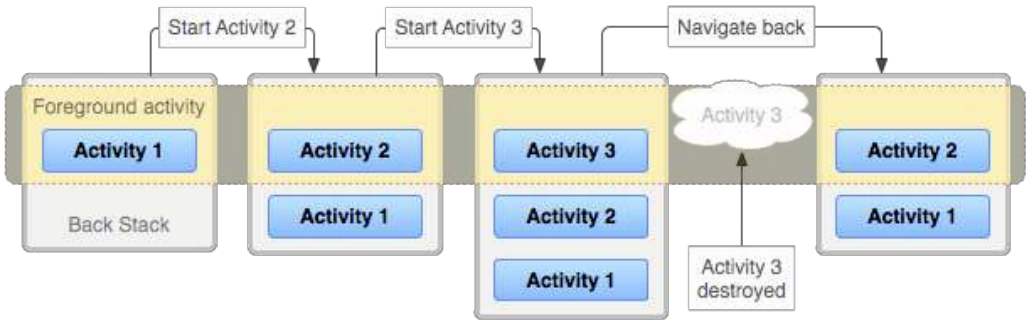
**Activity stack**

An android application usually contains lots of activities. We can navigate over these activities. When a user starts an activity, android OS pushes that into a stack. If user starts another activity then first activity goes down and newly started activity is added to the top of the stack. When user pushes back button, then top activity is destroyed and the activity which is below the top activity is resumed.

**Example:**

For example, a user receives a message on android phone. Now consider the actions performed by the user. By tracking those actions, we can understand how activity stack works. The user views the list of messages in inbox is activity 1. The user opens a message for reading is Activity
2. If the user replies to that message that becomes activity 3. If the user presses back button, then he gets activity 2 again. When he gets activity 2 again, the activity 3 is destroyed. Together these groups of activities form a task.

A task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the *back stack*), in the order in which each activity is opened.




A representation of how each new activity in a task adds an item to the back stack. When the user presses the *Back* button, the current activity is destroyed and the previous activity resumes.

## 2. Life Cycle of an Activity

An android activity has a lifecycle during performing a few things. Why is it required? Let's understand it with an example.

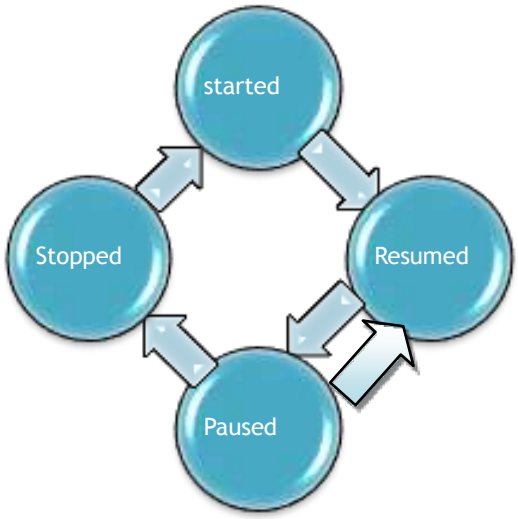
For example, a user is playing game in his mobile. During that time, he receives a call from someone. When playing game, if the user receives a call the game is paused and the calling window appears on top of the game screen. When the call ends, the user can resume the game at the point where he left. Here game is an application. When that application is interfered by any other activity or application, the state of the game application is saved.

How the game state is saved and resumed? There must be some technique to save the game state. To handle such situations android provides activity lifecycle. The life cycle of an activity is handled by a set of functions and they are invoked when something happens to the activity.

**Activity States**

An activity life cycle starts from creating an activity. A created activity, in a running application can be in one of the following three states:

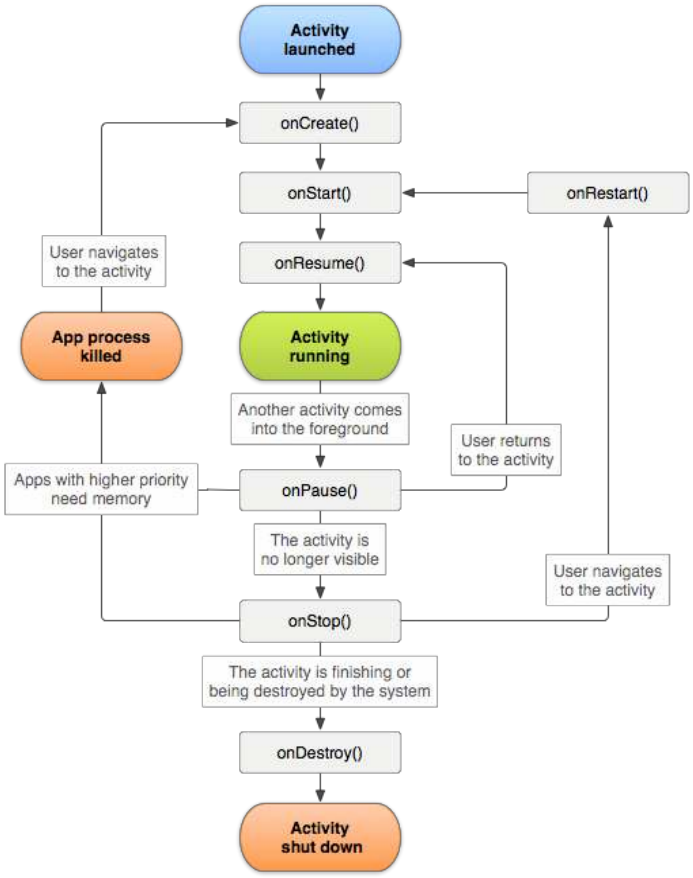
- Resumed – Activity is running or in active state, it is focused and the user can interact with it.
- Paused – Activity is not running or focused, it might be covered by another activity, or is transparent)
- Stopped – Activity is stopped or not visible.




When an activity goes from one state to another, the different life cycle methods are invoked where we can fill in with our code.

**Methods for Activity Life Cycle**

When an activity transitions from one state to another, set of callback methods are invoked for each state. How a set of call back methods for each state of an activity is invoked in its life cycle is given in the below picture.



Following are the available callback methods

- onCreate()
- onStart()
- onPause()
- onResume()
- onStop()
- onRestart()
- onDestroy()

**onCreate**

The onCreate() method is called when an activity is created for the first time. This method is invoked only once during the entire lifecycle of an activity. To create an activity, we must override the onCreate() method which is available in the class named Activity. Since this method is called only once at the beginning of an activity it can be used for initialization.

As the onCreate() method is called while creating an activity, the method for setting the activity layout setContentView() is always invoked inside of this method.

**onStart:**

The onStart() method is called after the creation of an activity and just before the activity becomes visible to the user. This onStart() method can be called from two places - after onRestart() and OnCreate(). i.e., after creating an activity to start it for a first time, or to restart an activity. This onStart() method can also be used to reset any data of an activity.

**onResume:**

The onResume() method is called when our activity comes into the foreground, from the paused state. We already discussed an example that receiving a call while playing game in a phone. In that example, when the received call is ended the game is resumed. During that time (when game is resumed), the game activity is on top of the activity stack, so that the activity is ready to interact with user. onResume() is a good place to update the screen with new results.

**onPause:**

The onPause() method is invoked, when a new activity comes on top of the existing activity. Typically anything that steals the user away from an activity will result in calling onPause() method. The onPause() method can have the code for releasing resources, saving the application data, stopping background threads etc. It is always guaranteed that whenever the activity is becoming invisible or partially invisible, onPause() will be called. But once onPause() is called, android reserves the right to kill our activity at any point. Hence we should not be relying on receiving any further events.

**onStop:**

The onStop() method gets called when an activity finishes its work or stopped by the user. We can use this method to shut down when we need to create time intensive or CPU intensive operations.

**onRestart:**

The onRestart() method is similar to onCreate(), but onRestart() is called only after onStop(). Through this method in an application, we can understand that whether the application is starting a fresh or getting restarted. When an activity invokes onRestart(), the activity state is stored and variables are reinitialised.

**onDestroy:**

This method is invoked at the last stage of an activity life cycle. When an activity is killed, the onDestroy() method is invoked. For example consider that when the user presses back button on any activity, the foreground activity gets destroyed and control will be returned to the previous activity.

Even though, we use always onPause() and onStop() to clean up resources, the onDestory() allows our application to have another chance to do that before it exits. But there is no guarantee that onDestroy() will be called. It will be called only when the system is low on resources or user press the back button or if we use finish () explicitly in our code.

**Process Lifecycle**

The Android system attempts to keep application process around for as long as possible, but eventually will need to remove old processes when memory runs low. As described in Activity Lifecycle, the decision about which process to remove is intimately tied to the state of the user's interaction with it. In general, there are four states a process can be in based on the activities running in it, listed here in order of importance. The system will kill less important processes (the last ones) before it resorts to killing more important processes (the first ones).

- The foreground activity (the activity at the top of the screen that the user is currently interacting with) is considered the most important. Its process will only be killed as a last resort, if it uses more memory than is available on the device. Generally at this point the device has reached a memory paging state, so this is required in order to keep the user interface responsive.

- A visible activity (an activity that is visible to the user but not in the foreground, such as one sitting behind a foreground dialog) is considered extremely important and will not be killed unless that is required to keep the foreground activity running.

- A background activity (an activity that is not visible to the user and has been paused) is no longer critical, so the system may safely kill its process to reclaim memory for other foreground or visible processes. If its process needs to be killed, when the user navigates back to the activity (making it visible on the screen again), its onCreate(Bundle) method will be called with the savedInstanceState it had previously supplied in onSaveInstanceState(Bundle) so that it can restart itself in the same state as the user last left it.

- An empty process is one hosting no activities or other application components (such as Service or BroadcastReceiver classes). These are killed very quickly by the system as memory becomes low. For this reason, any background operation you do outside of an activity must be executed in the context of an activity BroadcastReceiver or Service to ensure that the system knows it needs to keep your process around.

Sometimes an Activity may need to do a long-running operation that exists independently of the activity lifecycle itself. An example may be a camera application that allows you to upload a picture to a web site. The upload may take a long time, and the application should allow the user to leave the application will it is executing. To accomplish this, your Activity should start a Service in which the upload takes place. This allows the system to properly prioritize your process (considering it to be more important than other non-visible applications) for the duration of the upload, independent of whether the original activity is paused, stopped, or finished.

## 3. Creating an Activity

Creating an activity class is as simple as creating a class in java. It simply involves creating a class and overriding a method. To create an activity, we need to extend the Activity class which is available in the android.app package. The onCreate() method must be overridden in order to create a new activity. The example code gives an understanding of creating an activity class and event handling in android. In an android project,

- Create an xml file under layout folder which is available in app/src/main/res
- Create an activity class under app/src/main/java folder

**Layout XML file**

By default, an xml file named activity_main.xml is available under res/layout folder. We can create user interface through two ways either by xml file or by java code. But the previous technique is preferred i.e., through xml file. Using the palette available in android studio, design the user interface. When controls are created, automatically code is placed in the xml file as shown below.

```
    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_marginTop="37dp"
        android:layout_centerHorizontal="true"
        android:height="30dp" android:width="50dp"
        android:background="#ff76ffa6" />
```

For each control an id is given which will be used as a reference to that control in code. Here the attribute android:id contains the id for the control EditText.

The code for the controls is placed inside opening and closing layout tag. The layout tag looks like as follows.

```
<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" android:paddingLeft="@dimen/
activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"

---------code for controls goes here                    </RelativeLayout>
```

There different types of layouts available such as relative layout, linear layout, grid layout etc. The layouts are covered in detail in the user interface chapter. After creating the layout xml file, we need to create activity class.

**Activity class**

We have created an activity class called MainActivity by extending the Activity class. Using this activity we are going to perform addition of two numbers. To add two numbers we will get input through text boxes and one button to perform the action. Finally to display the result one text box or label we can have. The Button, EditText and TextView controls are available in android.widget package.

**Example**

```
public class MainActivity extends Activity
{
        Button button; EditText
        editText; EditText
        editText2; TextView
        textView4;

    @Override
        protected void onCreate(Bundle savedInstanceState)
                { super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main_activity2);
```

The onCreate() method is overridden. If we notice that the onCreate() method is having a parameter Bundle type. Bundle is typically used to store the state of the Activity. Consider the example screen rotation, during which the Activity gets killed or paused and onCreate() is called again. We can determine if the Activity was already there, using Bundle so that you do not have to create the Activity again.

To more elaborate, consider the screen rotation example. If a user has already filled some of the fields and suddenly rotates his screen, the values filled in will be lost. Using Bundle, android retains the values of these fields and re-populates the data after rotation automatically. The value of Bundle will always be null when Activity is getting created for the first time. Always the

setContentView() method is called to display the user interface (to set the layout and controls for activity). This method receives the layout xml file is as a parameter R.layout. activity_main.

Now the controls in the code must be linked to the controls in the xml file. In order to do that, we call findViewById() method. This method is used for identifying the controls in the layout file by their attribute named id. Since the return type of finViewById() does not match with the controls type, they have type casted. The R is a class which stands for resources. It is automatically generated and cannot be modified.

```
button = (Button) findViewById(R.id.button); editText = (EditText)
findViewById(R.id.editText); editText2 = (EditText) findViewById(R.id.editText2);
textView4 = (TextView) findViewById(R.id.textView4);
```

After creating the UI, it is time to make work them. To perform event handling we need to implement the listener interface and add the listener to the button control.

```
button.setOnClickListener(this);
```

Now if any click event happens on the button, the button is notified by the listener. To handle the

```
public void onClick(View view) { int a,b,c;
    a=Integer.parseInt(editText.getText().toString());
    b=Integer.parseInt(editText2.getText().toString()); c=a+b;
    textView4.setText("The result is "+c);
}
```

event, the code is written inside the event handler which is shown below.

The complete code is given below.

**Example**

**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"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@+id/
        editText" android:layout_marginTop="37dp"
        android:layout_centerHorizontal="true"
        android:height="30dp"
        android:width="50dp"
        android:background="#ff76ffa6" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="Enter
            the number1" android:id="@+id/textView2"
            android:layout_alignTop="@+id/editText"
```

```
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignBottom="@+id/editText" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Enter
        the number2" android:id="@+id/textView3"
        android:layout_below="@+id/textView2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="31dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@
        +id/editText2" android:layout_alignTop="@+id/
        textView3" android:layout_alignLeft="@+id/editText"
        android:layout_alignStart="@+id/editText"
        android:width="50dp" android:height="30dp"
        android:background="#ff6fffab" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add" android:id="@+id/button"
        android:layout_below="@+id/editText2"
        android:layout_alignRight="@+id/editText2"
        android:layout_alignEnd="@+id/editText2"
        android:layout_marginTop="45dp" />

    <TextView
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Result"
        android:id="@+id/textView4" android:layout_below="@+id/button"
        android:layout_alignRight="@+id/editText2" android:layout_alignEnd="@+id/
        editText2" android:layout_marginTop="85dp" />

    </RelativeLayout>
```

**MainActivity.java**

```
public class MainActivity extends Activity implements OnClickListener
{
        Button button; EditText
        editText; EditText
        editText2; TextView
        textView4;

        @Override
        protected void onCreate(Bundle savedInstanceState)
                { super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                button = (Button) findViewById(R.id.button); editText = (EditText)
                findViewById(R.id.editText); editText2 = (EditText)
                findViewById(R.id.editText2); textView4 = (TextView)
                findViewById(R.id.textView4); button.setOnClickListener(this);
        }
                        public void onClick(View view) { int a,b,c;
                                a=Integer.parseInt(editText.getText().toString());
                                b=Integer.parseInt(editText2.getText().toString()); c=a+b;
                                textView4.setText("The result is "+c);
        }
}
```

**Output**


## 2.2 Intents

Actually intents are not one of android application components; but used for activating components in Android. It is the part of core message system in android. It defines a message to activate the target component. For example, if we want to invoke a new activity from our current activity, then we need to fire an intent specifying the new activity. Further if we want to start another application from our activity, then also we require intent.

Intent is a messaging object that can be used to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

- Activity
- Service
- Broadcast

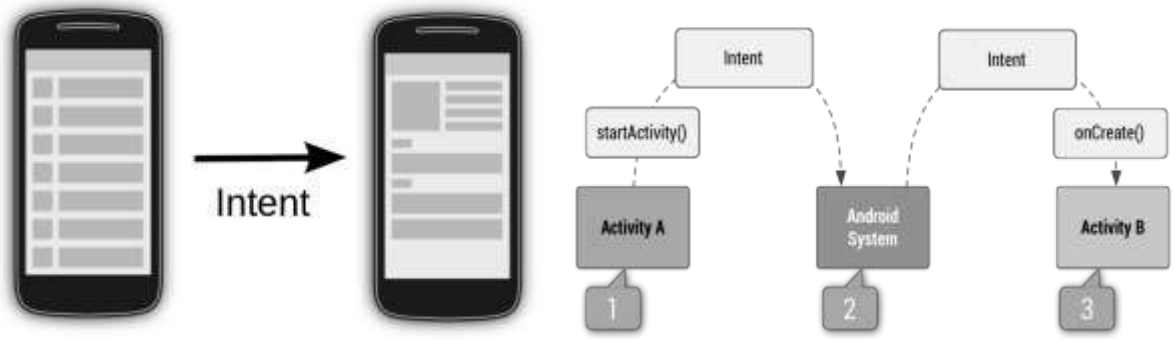
### Starting activities

We may want to start another activity from the existing one either as a new activity on top of the activity stack or as a result back from the previous one. For example, if we start an activity which allows the user to pick a person from a list of contacts, it returns the person that was selected when selection completed i.e end of previous activity.

- startActivity(Intent)
- startActivityForResult(Intent, int)

The startActivity(Intent) method is used to start a new activity, which will be placed at the top of the activity stack. It takes a single argument, an Intent, which describes the activity to be executed. Sometimes we may want to get a result back from an activity when it ends. To accomplish this, the startActivityForResult(Intent, int) method can be used. It is another form of startActivity() method with a parameter identifying the call. The result will come back through our onActivityResult(int, int, Intent) method.

To start an activity, the startActivity(intent) method will be invoked. This method is defined on the Context object which Activity extends.




The following code demonstrates how another activity is started via intent.

```
# Start the activity connect to the
# specified class

Intent i = new Intent(this, ActivityTwo.class); startActivity(i);
```

Intents are covered in detail later on; however we need to use Intents in Activities. So we will explore briefly the concept of Intents.

## Types of Intents

Intents have been classified into two types. They are

- Explicit Intents
- Implicit Intents

1) **Explicit Intents:**

Explicit intent is called for internal communication of an application. It is being invoked by mentioning the target component name. The component or activity can be specified which should be active on receiving the intent. For this reason mostly it is used for intra application communication. The following code describes the way of creating an explicit intent. While creating explicit intent, the target activity is specified so that the android system will invoke the activity.

```
Intent i = new Intent (this, Activity2.class); i.putExtra("one", "This is value
one for Activity2 "); i.putExtra("two", "This is value two for Activity2");
```

**Example**

Activity1

```
int a,b,c;
                    a=Integer.parseInt(editText.getText().toString());
                    b=Integer.parseInt(editText2.getText().toString()); c=a+b;
Intent i=new Intent(this, MainActivity2.class);

i.putExtra("result", c);
startActivity(i);
```

Now the addition of two numbers program has been modified in order to use intent. In the first activity, inside the onClick() method intent object is created. While creating intent object the target component is specified that should be get activated. Here the MainActivity2 is the target activity that will be invoked by this intent. Since intents are part of the core message system of android, data is passed to the second activity 'MainActivity2'. i.e. input numbers are read in the first activity, when the button is clicked intent object is created and the result is passed through the intent object. It will activate the target activity. We will get the result in the second activity. The code for second activity is given below.

Activity2

```
Bundle data=getIntent().getExtras();
        if(data==null){
            return;                    }
       int myData=data.getInt("result");
```

The second activity is activated by the intent object. Also the second activity receives data or message from the first activity through intent object. The Bundle class is used to save the activity state. So data from the intent object is read with the help of Bundle. The getIntent() method will retrieve data from the intent object.

Intent can have the following information.

a) **Component name** - The name of the component to start. This is optional, but it's the critical piece of information that makes an intent explicit, meaning that the intent should be delivered only to the app component defined by the component name. Without a component name, the intent is implicit and the system decides which component should receive the intent based on the other intent information (such as the action, data, and category).
b) **Action** - A string that specifies the generic action to perform (such as view or pick). Some common actions for starting an activityare:
    - ACTION_VIEW - This action is used in an intent with startActivity() when it has some information that an activity can show to the user, such as a photo to view in a gallery app, or an address to view in a map app.
    - ACTION_SEND - Also known as the "share" intent, this intent is used with startActivity() when some data that the user can share through another app, such as an email app or social sharing app.
c) **Data** - The URI (a Uri object) that references the data to be acted on and/or the MIME type of that data. The type of data supplied is generally dictated by the intent's action. For example, if the action is ACTION_EDIT, the data should contain the URI of the document to edit.
d) **Categories** - A string containing additional information about the kind of component that should handle the intent.

    - **Extras** - Key-value pairs that carry additional information required to accomplish the requested action.
    - **Flags** - Flags defined in the Intent class that function as metadata for the intent. The flags may instruct the Android system how to launch an activity.

2) **Implicit Intents:**


When implicit intents are used, a message is sent to the android system to find an appropriate activity to respond to the intent. For example, to share a video, we can use intent. The video can be shared through any type of external application. To do this we can use intent.

When intent is received, android system will invoke an activity which is capable of sending video. If there is more than one activity is capable of receiving the intent, the android system will present a chooser so that the user can select which activity or application should handle it. The following sample code shows the use of implicit intents.

```
Intent intent = new Intent(Intent.ACTION_VIEW,
                                    Uri.parse("http://www.ebookfrenzy.com"));

            startActivity(intent);
```

The above code sends an url to open a web page. The Android system identifies the appropriate target component and complete the action. Here the Android system will open a browser to view the given url of a web page. The intent did not mention the target component to activate.

## 2.3 GUI Components

### Basics of User Interface

App's user interface is everything that the user can see and interact with. Android provides a variety of pre-build UI components such as structured layout objects and UI controls that allow us to build the graphical user interface for our app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

All user interface elements in an android app are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface. All UI controls have attributes and id. Attributes are used to specify values for the property of a UI control. The id for a UI control is used for referring the controls.

**Attributes**

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class. Some are common to all View objects, because they are inherited from the root View class (like the id attribute). And, other attributes are considered "layout parameters" which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.

**ID**

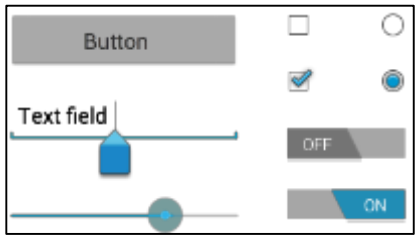
Any View object may have an integer ID associated with it, to uniquely identify the View within the tree. When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute. This is an XML attribute common to all View objects (defined by the View class) and you will use it very often. The syntax for an ID, inside an XML tag is:

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

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 (inthe R.java file). There are a number of other ID resources that are offered by the Android framework. When referencing an Android resource ID, you do not need the plus-symbol, but must add the android package namespace, like so

## Input Controls

Input controls are the interactive components in our app's user interface. Android provides a wide variety of controls we can use in our UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons etc.




## Common Controls

Android provides more controls but only few are listed here. To explore other controls browse the android.widget package. If our app requires a specific kind of input control, we can build our own custom components.

| Control Type | Description | Related Classes |
|---|---|---|
| Button | A push-button that can be pressed, or clicked, by the user to perform an action. | Button |
| Text field | An editable text field. You can use theAutoCompleteTextView widget to create a text entry widget that provides auto-complete suggestions | EditText,AutoCompleteTextView |
| Checkbox | An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive. | CheckBox |
| Radio button | Similar to checkboxes, except that only one option can be selected in the group. | RadioGroup, RadioButton |
| Toggle button | An on/off button with a light indicator. | ToggleButton |
| Spinner | A drop-down list that allows users to select one value from a set. | Spinner |
| Pickers | A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use aDatePickercode> widget to enter the values for the date (month, day, year) or<br>a TimePicker widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale. | DatePicker,TimePicker |

## Working with Controls

The commonly used controls have been discussed. There are various controls available under android.widget package. Few controls have been covered here.

**EdiText**

We can add a text box control to our app by adding EditText control called EditTextName in the layout file. This EditText control called EditTextEmail acts as a form field for the user's email address. The following code shows the edit text control added to the layout xml file. The edit text control has a property called android:inputType . It provides advantage for users to set the type of input we want to receive in the text box. The value for the input type property can be any of the valid property values such as text, number, password, email etc.

```
<EditText
        android:id="@+id/EditTextEmail"
        android:layout_height="wrap_content"
        android:hint="@string/feedbackemail"
        android:inputType="textEmailAddress"
        android:layout_width="fill_parent">
```

To read value from the text box the getText() method can be used. The following code illustrates the use of getText() method.

```
final EditText emailField = (EditText)
findViewById(R.id.EditTextEmail);
String email = emailField.getText().toString();
```

**Radio Buttton**

To insert a radio button in a layout xml file we need to create radio button group. Under the radio group we can add as many as radio buttons we want.

```
<RadioGroup
                        android:id="@+id/radioGender"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content" >

                        <RadioButton
                                android:id="@+id/genderMale"
                                android:layout_width="wrap_content"
                                android:text="Male"
                                android:checked="true" />

                        <RadioButton
                                android:id="@+id/genderFemale"
                                android:layout_width="wrap_content"
                                android:layout_height="wrap_content"
                                android:text="Female" />

                </RadioGroup>
```

To identify the selected radio button, we can use the getCheckedRadioButtonId() method. It returns the id of the radio button which is selected. With the help of the id, we can read the value of the radio button using getText() method.

```
radioSexGroup = (RadioGroup) findViewById(R.id.radioGender); btnDisplay = (Button)
findViewById(R.id.btnDisplay)

    // get selected radio button from radioGroup
  int selectedId = radioSexGroup.getCheckedRadioButtonId();

// find the radiobutton by returned id
radioSexButton = (RadioButton) findViewById(selectedId);

  //read value from radio button
radioSexButton.getText()
```

**Check Box**

The CheckBox control can be added to layout file as given below. Similar to other controls we need to provide id for the control which will be used in the activity file to refer it. To display an option to the user besides check box, directly we can hardcode the string or pass the string value through strings xml file. The following code snippet show sample checkbox control added in a layout file.

```
<CheckBox
        android:id="@+id/CheckBoxResponse"
        android:layout_height="wrap_content"
        android:text="@string/feedbackresponse"
        android:layout_width="fill_parent">
```

After completing the design of the checkbox control, we need to read data from the user. Through code, we need to identify which check box was selected by the user or whether a check box is selected or not. Sample code for reading data from a check box control is given below.

```
final CheckBox responseCheckbox = (CheckBox)
findViewById(R.id.CheckBoxResponse);
boolean bRequiresResponse = responseCheckbox.isChecked();
```

The isChecked() method is used to find whether a check box is selected by the user or not selected.

**Spinner Control**

To add a Spinner control we need to add the spiner control to the layout file. In the following sample code the Spinner control called SpinnerFeedbackType allows the user to select the type of feedback from a fixed list of options (Praise, Gripe, Suggestion, or Bug).

First, we need to define these choices as individual string resources in the strings.xml resource file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!--Other string resources also defined in this file… -->
<string name="feedbacktype1">Praise</string>
<string name="feedbacktype2">Gripe</string>
<string name="feedbacktype3">Suggestion</string>
<string name="feedbacktype4">Bug</string>
</resources>
```

Next, we need to create a string array resource using the individual string resources as follows in /res/values/arrays.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="feedbacktypelist">
<item>@string/feedbacktype1</item>
<item>@string/feedbacktype2</item>
<item>@string/feedbacktype3</item>
<item>@string/feedbacktype4</item>
</string-array>
</resources>
```

Now we can configure the Spinner control in the layout. We can begin by supplying the prompt attribute, which will provide a helpful string at the top of the Spinner control. Next, we need to specify the list of string choices using the entries attribute—specifically, set the entries attribute to the string array we just defined: @array/feedbacktypelist.

```
<Spinner
    android:id="@+id/SpinnerFeedbackType"
    android:layout_height="wrap_content" android:prompt="@string/
    feedbacktype" android:layout_width="fill_parent"
    android:entries="@array/feedbacktypelist">
</Spinner>
```

After completing the design of UI control, we need to write the code to read data from the control. The getSelectedItem() method is used to read the data from this spinner control.

```
final Spinner feedbackSpinner = (Spinner)
findViewById(R.id.SpinnerFeedbackType);
String feedbackType = feedbackSpinner.getSelectedItem().toString();
```

## Android Studio Project SubDirectories:

**1. > Manifests**

> AndroidManifest.xml

**2. > java**

> com.example.kamarajios34.guicomponents

> MainActivity

> com.example.kamarajios34.guicomponents (androidTest)

>ApplicationTest

**3. > res**

> drawable
> layout
> activity_main.xml
> mipmap
> ic_launcher.png
> ic_launcher.png (hdpi)
> ic_launcher.png (mdpi)
> ic_launcher.png (xhdpi)
> ic_launcher.png (xxhdpi)
> ic_launcher.png (xxxhdpi)

**4. > values**

> colors.xml
> dimens.xml
> strings.xml
> styles.xml

**5. > Gradle Scripts**

# 2.4 Running the GUI Components, Fonts, Colours App

## GUI Components and colours

The following program illustrates the GUI Components and Colours for Android App

### Step 1: Design

1. Open the actual Project folder(app) in Android Studio IDE
2. Click res directory -> layout -> activity_main.xml -> Design
3. Insert the GUI components to Design view in activity_main.xml
4. Enter the id for each component

**Step 2:** Open res directory -> layout -> activity_main.xml and add following code

```
        <?xml version="1.0" encoding="utf-8"?>
<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" android:paddingLeft="@dimen/
activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="GUI Components"
      android:id="@+id/t1"
      android:layout_alignParentTop="true"
      android:layout_alignParentEnd="false"
      android:layout_alignParentStart="false"
      android:autoText="false"
      android:minHeight="40dp"
      android:textStyle="bold"
      android:textSize="30dp"
      android:textIsSelectable="false"
      android:textAlignment="center"
      android:textColor="@color/accent_material_dark" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Green"
      android:id="@+id/b1"
      android:layout_alignTop="@+id/b2"
      android:layout_alignParentStart="true" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Red"
      android:id="@+id/b2"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />
```

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Blue"
        android:id="@+id/b3"
        android:layout_alignTop="@+id/b2"
        android:layout_alignEnd="@+id/t1" />

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/img1"
        android:maxHeight="70dp"
        android:layout_alignParentBottom="true"
        android:src="@mipmap/ic_launcher"
        android:layout_alignParentEnd="true"
        android:layout_below="@+id/b1" />

    <ImageButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/imgb1"
        android:maxHeight="10dp"
        android:maxWidth="10dp"
        android:longClickable="false"
        android:visibility="visible"
        android:minHeight="10dp"
        android:contentDescription="CLICK BUTTON"
        android:layout_gravity="left|top|bottom|center|right"
        android:layout_above="@+id/b1"
        android:layout_alignEnd="@+id/b3"
        style="@style/AlertDialog.AppCompat.Light"
        android:background="#dd2e2e"
        android:layout_marginLeft="“@dimen
        abc_action_bar_subtitle_bottom_margin_material"
        android:layout_marginRight="@dimen/
abc_action_bar_subtitle_bottom_margin_material"
        android:layout_marginBottom="@dimen/
abc_action_bar_subtitle_bottom_margin_material" />
```

```
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click RED button"
        android:id="@+id/textView"
        android:layout_above="@+id/imgb1"
        android:layout_alignStart="@+id/imgb1" />

    <ToggleButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="New ToggleButton"
        android:id="@+id/toggleButton"
        android:layout_below="@+id/t1"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

**Step 3:** Open your Android studio project folder (e.g. Project name: GUI Components ) —> Click app -> src -> main -> res ->  drawable -> add *.png file.

**Step 4:** Open java -> MainActivity.java and add following code

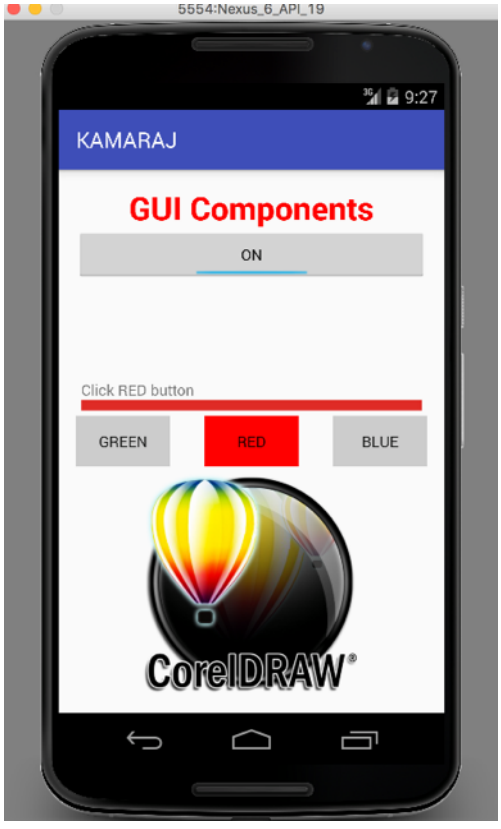
```
package com.example.kamarajios34.guicomponents;

import android.graphics.Color;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.media.Image;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.TextView;
import org.w3c.dom.Text;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
```

```
public class MainActivity extends AppCompatActivity {
    TextView text;
    Button bu1,bu2,bu3;
    ImageView image1;
    ImageButton image2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main)
        bu1=(Button) findViewById(R.id.b1);
        bu2=(Button) findViewById(R.id.b2);
        bu3=(Button) findViewById(R.id.b3);
        text=(TextView) findViewById(R.id.t1);
        image1=(ImageView)findViewById(R.id.img1);
        image2=(ImageButton)findViewById(R.id.imgb1);
        image2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                image1.setImageResource(R.drawable.corel);

            }
        });

bu1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                    bu1.setBackgroundColor(Color.GREEN);
                    text.setTextColor(Color.GREEN);
                    bu2.setBackgroundColor(Color.LTGRAY);
                    bu3.setBackgroundColor(Color. LTGRAY);
            }});

  bu2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                bu2.setBackgroundColor(Color.RED);
                bu3.setBackgroundColor(Color.LTGRAY);
                bu1.setBackgroundColor(Color.LTGRAY);
                text.setTextColor(Color.RED);
            }});
        bu3.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                bu3.setBackgroundColor(Color.BLUE);
                text.setTextColor(Color.BLUE);
                bu2.setBackgroundColor(Color.LTGRAY);
                bu1.setBackgroundColor(Color.LTGRAY);
```

```
    });
}}
```

**Step 5:** The output of the above code is as follows. As we have discussed, GUI Components and Colours App is easy.





## 2.4 Running the Fonts App

### Step 1: Design

1. Open the  actual Project folder(app) in Android Studio IDE
2. Click  res directory -> layout  -> activity_main.xml ->  Design
3. Insert the GUI components to Design  view in activity_main.xml
4. Enter the id for each component


**Step 2:** Create assets sub directory in res

Right click res -> New -> Folder -> New Assets Folder
Right click assets -> New -> Directory -> Enter the directory name (fonts)
paste the font types in fonts directory

**Step 3:** Create an activity file

**MainActivity.java**

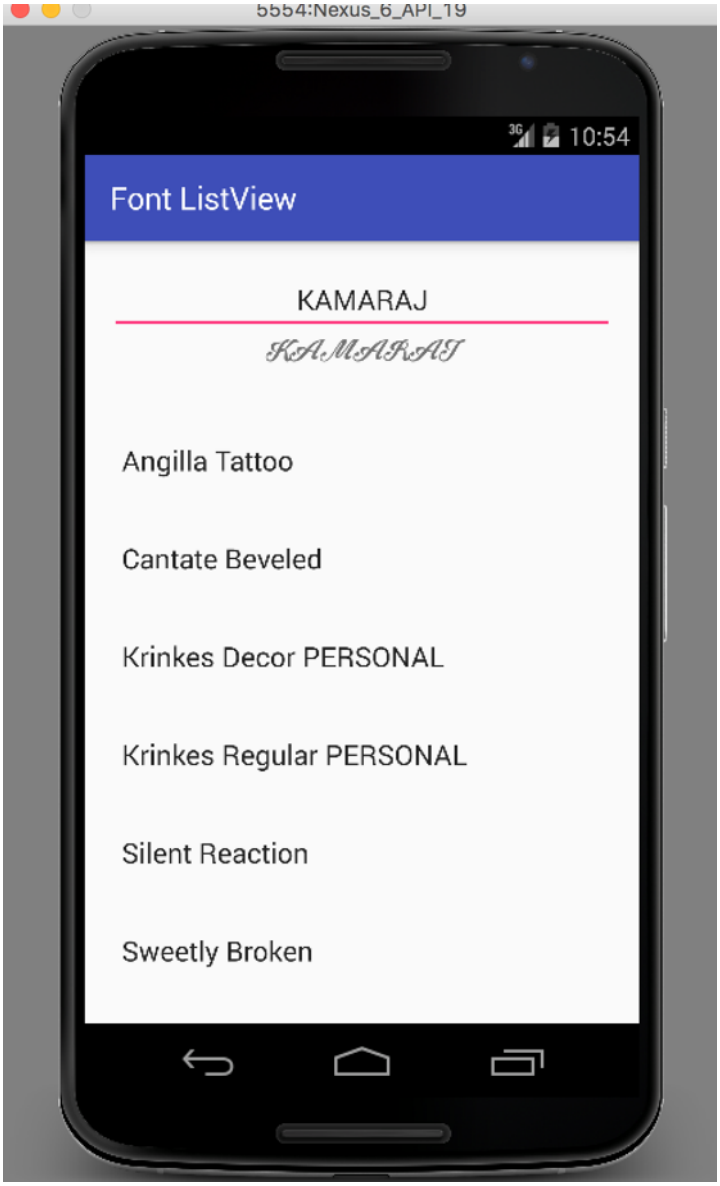
```
package com.example.kamarajios34.fontlistview;

import android.graphics.Typeface;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
```

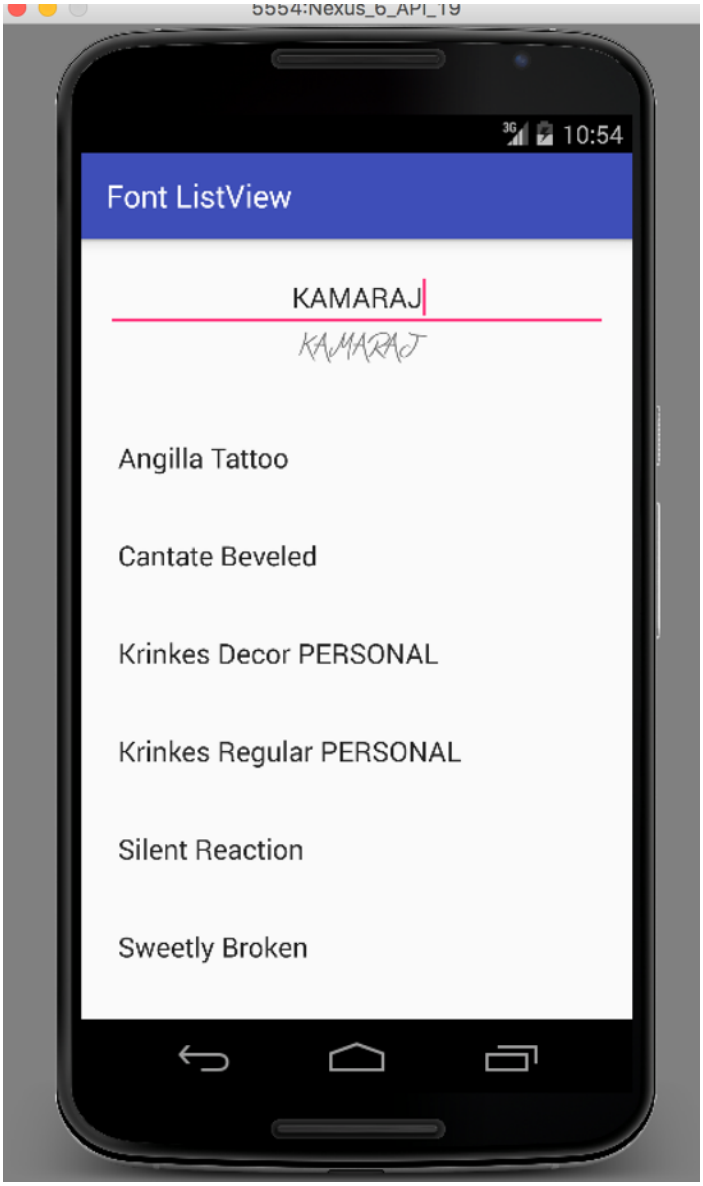
```
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
   ListView listview;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      final TextView txt=(TextView)findViewById(R.id.textView);
      final EditText e1=(EditText)findViewById(R.id.edit1);
            listview=(ListView)findViewById(R.id.list);
      String[] values = new String[] {

            "Angilla Tattoo",
            "Cantate Beveled",
            "Krinkes Decor PERSONAL",
            "Krinkes Regular PERSONAL",
            "Silent Reaction",
            "Sweetly Broken",
            "Xerox Sans Serif Narrow Bold",
            "Xacto Blade"
      };


      ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line,values);



      listview.setAdapter(adapter);

     listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long
id)
        {

           String itemValue = (String) listview.getItemAtPosition(position);
 if(itemValue.equals("Angilla Tattoo"))
            {
              txt.setText(e1.getText().toString());
              String fontPath="fonts/AngillaTattoo_PERSONAL_USE_ONLY.ttf";
             Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
```

```
   txt.setTypeface(tf);
 }

if(itemValue.equals("Cantate Beveled"))
 {
    txt.setText(e1.getText());
    String fontPath="fonts/Cantate Beveled.ttf";
    Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
    txt.setTypeface(tf);
 }

 if(itemValue.equals("Krinkes Decor PERSONAL"))
 {
    txt.setText(e1.getText());
    String fontPath="fonts/KrinkesDecorPERSONAL.ttf";
    Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
    txt.setTypeface(tf);
 }

 if(itemValue.equals("Krinkes Regular PERSONAL"))
 {
    txt.setText(e1.getText());
    String fontPath="fonts/KrinkesRegularPERSONAL.ttf";
    Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
    txt.setTypeface(tf);
 }

 if(itemValue.equals("Silent Reaction"))
 {
    txt.setText(e1.getText());
    String fontPath="fonts/Silent Reaction.ttf";
    Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
    txt.setTypeface(tf);
 }

 if(itemValue.equals("Sweetly Broken"))

 {
    txt.setText(e1.getText());
    String fontPath="fonts/Sweetly Broken.ttf";
    Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
    txt.setTypeface(tf);
 }

 if(itemValue.equals("Xerox Sans Serif Narrow Bold"))
 {
    txt.setText(e1.getText());
    String fontPath="fonts/Xerox Sans Serif Narrow Bold.ttf";
```

```
            Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
            txt.setTypeface(tf);
        }

        if(itemValue.equals("Xacto Blade"))
        {
            txt.setText(e1.getText());
            String fontPath="fonts/Xacto Blade.ttf";
            Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
            txt.setTypeface(tf);
        }}
});}}
```

**Step 4:** Output



# 3. Layout Managers and Event Listeners

## Objectives

- Layouts
- Event Listeners
- Menus

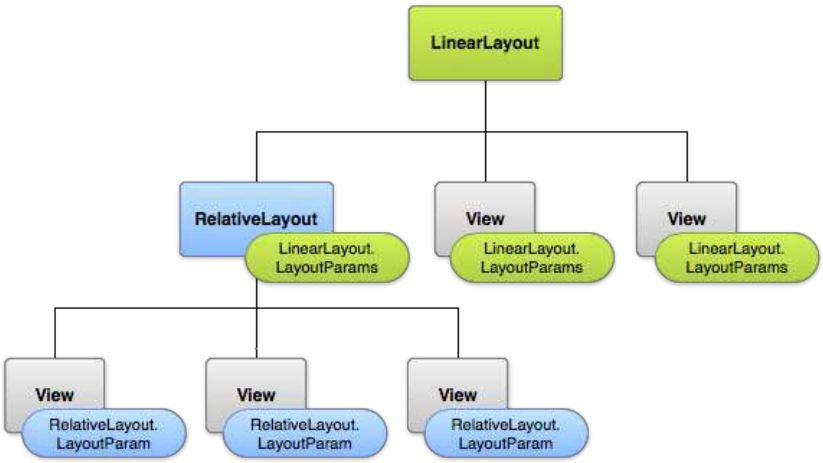
## 3.1 Layout

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.

Layout can be declared 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.
- **Instantiate layout elements at runtime**
  Our application can create View and ViewGroup objects (and manipulate their properties) programmatically.

XML layout attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which it resides. Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group. As given in the below picture, the parent view group defines layout parameters for each child view (including the child view group).




Visualization of a view hierarchy with layout parameters associated with each view.

**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.

| S.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 they are other attributes which are specific to that layout. Few attributes of layouts are given below.

| Attribute | Descriptio |
|---|---|
| android:id | This is the ID which uniquely identifies the view. |
| android:layout_width | This is the width of the layout. |
| android:layout_height | This is the height of the layout |
| android:layout_weight | This specifies how much of the extra space in the layout should be allocated to the View |
| android:layout_x | This specifies the x-coordinate of the layout. |
| android:layout_y | This specifies the y-coordinate of the layout. |
| android:paddingLeft | This is the left padding filled for the layout. |
| android:paddingRight | This is the right padding filled for the layout. |
| android:paddingTop | This is the top padding filled for the layout. |
| android:paddingBottom | This is the bottom padding filled for the layout. |

**Input Events and Event Handling**

Events are a useful way to collect data about user's interaction with interactive components of applications like button press, screen touch etc. The android framework maintains an event queue as first-in, first-out (FIFO) basis. We can capture these events in our program and take appropriate action as per requirements.

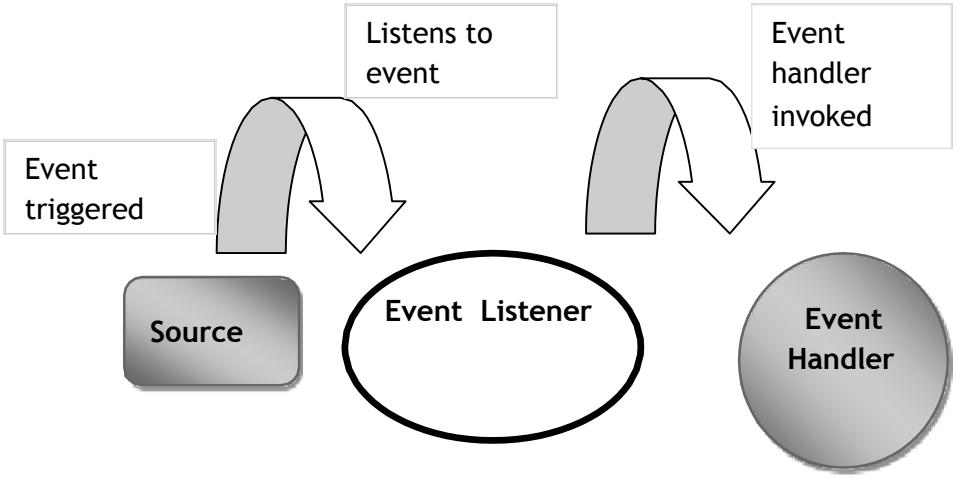
Events are actions performed by users. It can be in any form. These events are the inputs to the applications. When application interacts with user it receives input in the form events. For example an event may be a button click, key press etc.

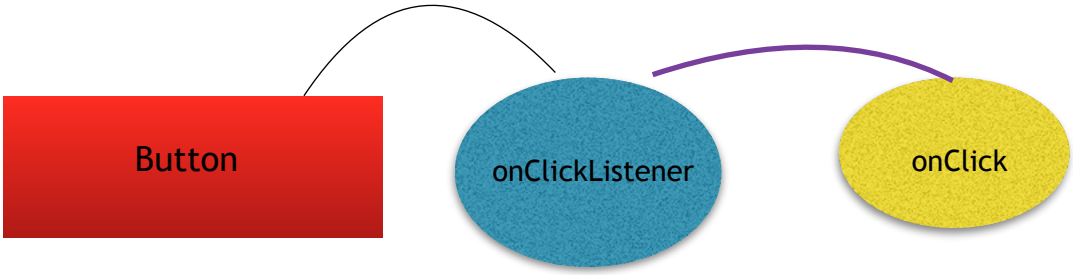
- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners 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 and event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

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. Though there are several tricky ways to register our event listener for any event, but we are going to discuss only 3 ways, out of which we can use based on the requirement or scenario.

- Using an Anonymous Inner Class
- Activity class implements the Listener interface.
- Using Layout file activity_main.xml to specify event handler directly.

**Event Handling Approach**






The event handling approach is to capture the events and take appropriate response based on the event type. Event handling is accomplished with the help of event listeners and event handlers. Event listeners listen to events and event handlers execute the appropriate code as response to the events.

When a control is created it must be registered with event listener in order to, notified by the listeners. When an event is triggered by users, the listeners are notified, and the listeners inform the event handlers that there is an event triggered. Then the event handler is executed, inside the event handler the code is written which is specifying what should happen for the event which was triggered.

Some of the event listeners and handlers are given below.

| **Event** | **Event Listener** | **Event Handler** | **Description** |
|---|---|---|---|
| Click | OnClickListener | onClick() | called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball. |
| Long click | OnLongClickListener | onLongClick() | This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second). |
| Navigation | OnFocusChangeListener | onFocusChange() | This is called when the user navigates onto or away from the item, using the navigation-keys or trackball. |
| Touch | OnTouchListener | onTouch() | This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item). |

| | | | |
|---|---|---|---|
| | OnMenuItemClickListener( ) | onMenuItem Click() | This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event. |

| | | | |
|---|---|---|---|
| | onCreateContext MenuItemListene r() | onCreateCont extMenu() | This is called when the context menu is being built(as the result of a sustained "long click) |

To know more kind of event handlers listeners we can refer official documentation for Android application development.

## 3.3 **Menus**

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, we should use the Menu APIs.

There are three types of application menus:

- Options Menu - The primary collection of menu items for an activity, which appears when the user touches the MENU button.
- Context Menu - A floating list of menu items that appears when the user touches and holds a view that's registered to provide a context menu.
- Submenu - A floating list of menu items that appears when the user touches a menu item that contains a nested menu.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. We will discuss how to create fundamental types of menus or action presentations on different versions of Android.


**Options menu**

The options menu is the primary collection of menu items for an activity. It's where we should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

If we are developing for Android 2.3 or lower, users can reveal the options menu panel by pressing the Menu button. On Android 3.0 and higher, items from the options menu are presented by the action bar as a combination of on-screen action items and overflow options. Beginning with Android 3.0, the Menu button is deprecated (some devices don't have one), so we should migrate toward using the action bar to provide access to actions and other options.

The action bar is a window feature that identifies the user location, and provides user actions and navigation modes. Use of action bar in an app, offers the users a familiar interface across applications that the system gracefully adapts for different screen configurations.


The action bar provides several key functions:

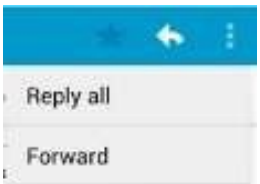
- Provides a dedicated space for giving our app an identity and indicating the user's location in the app.
- Makes important actions prominent and accessible in a predictable way (such as *Search*).
- Supports consistent navigation and view switching within apps (with tabs or drop-down lists).

**Context menu**

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

When developing for Android 3.0 and higher, we should instead use the contextual action mode to enable actions on selected content. This mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

**Popup menu**




A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should not directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in our activity.

The easiest way to create a menu is to define the menu in XML layout and then inflate a menu resource during the onCreateOptionsMenu() callback method.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/start_game" android:icon="@drawable/
        ic_start_game" android:title="@string/start_game" />
    <item android:id="@+id/game_help" android:icon="@drawable/
        ic_game_help" android:title="@string/game_help" />
</menu>
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu); return true;
}
```

When the user selects a menu item from the Options Menu the system calls the activity's onOptionsItemSelected() method.This method passes the MenuItem that the user selected. We can identify the menu item by calling getItemId(), which returns the unique ID for the menu item (defined by the android:id attribute in the menu resource or with an integer given to the add() method). We can match this ID against known menu items and perform the appropriate action. A menu group is a collection of menu items that share certain traits. With a group, you can:

- Show or hide all items with setGroupVisible()
- Enable or disable all items with setGroupEnabled()
- Specify whether all items are checkable with setGroupCheckable()

Dynamically Changing menu items at runtime

Once the activity is created, the onCreateOptionsMenu() method is called only once, as described above. The system keeps and re-uses the Menu defined in this method until the activity is destroyed. If we want to change the Options Menu any time after it's first created, we must override the onPrepareOptionsMenu() method. This passes the Menu object as it currently exists.

This example only deals with Options menu and Submenu only. This sample code helps to understand the Menu life cycle and how to create menus using XML layout and programmatically using code for adding and removing menus dynamically.

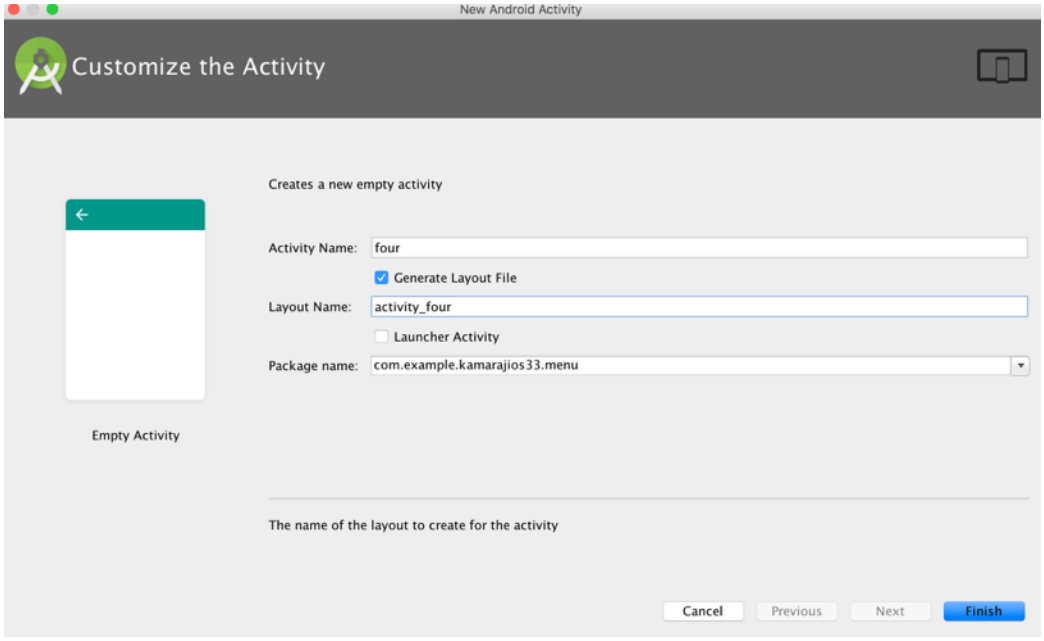
## 3.4 Running the Layout Managers and Event Click Listener App

**Step 1:** Create an android project using android studio

**Step 2:** After creating the project, open the res directory -> layout -> resource file named activity_main.xml

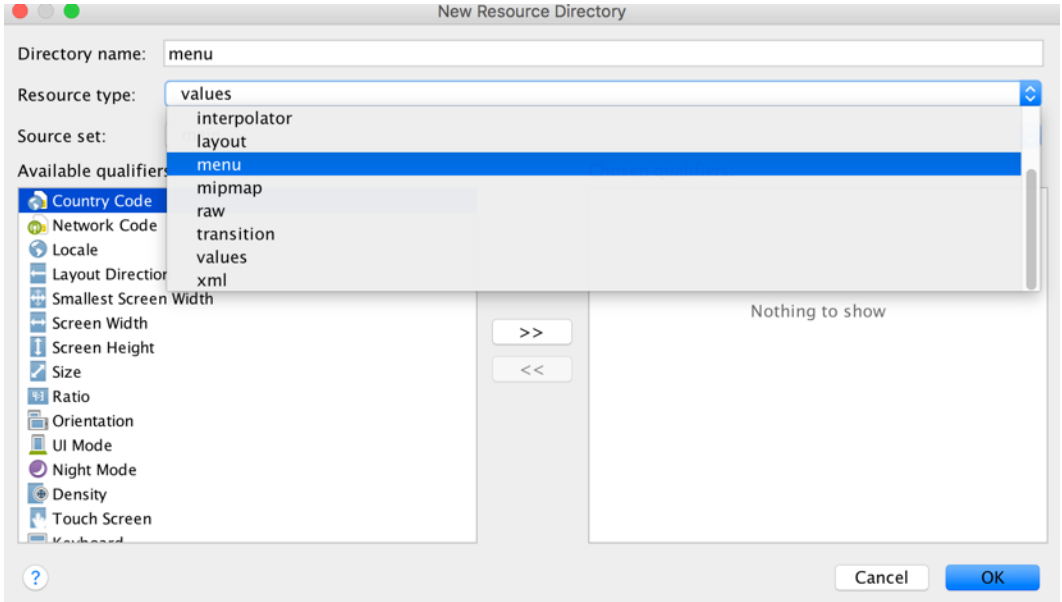
**Step 3:** Create two resource (*.xml) file named activity_second.xml, activity_third.xml & two activity (*.java) file named second.java and third.java file.

Right click res directory -> New -> Activity -> Empty Activity

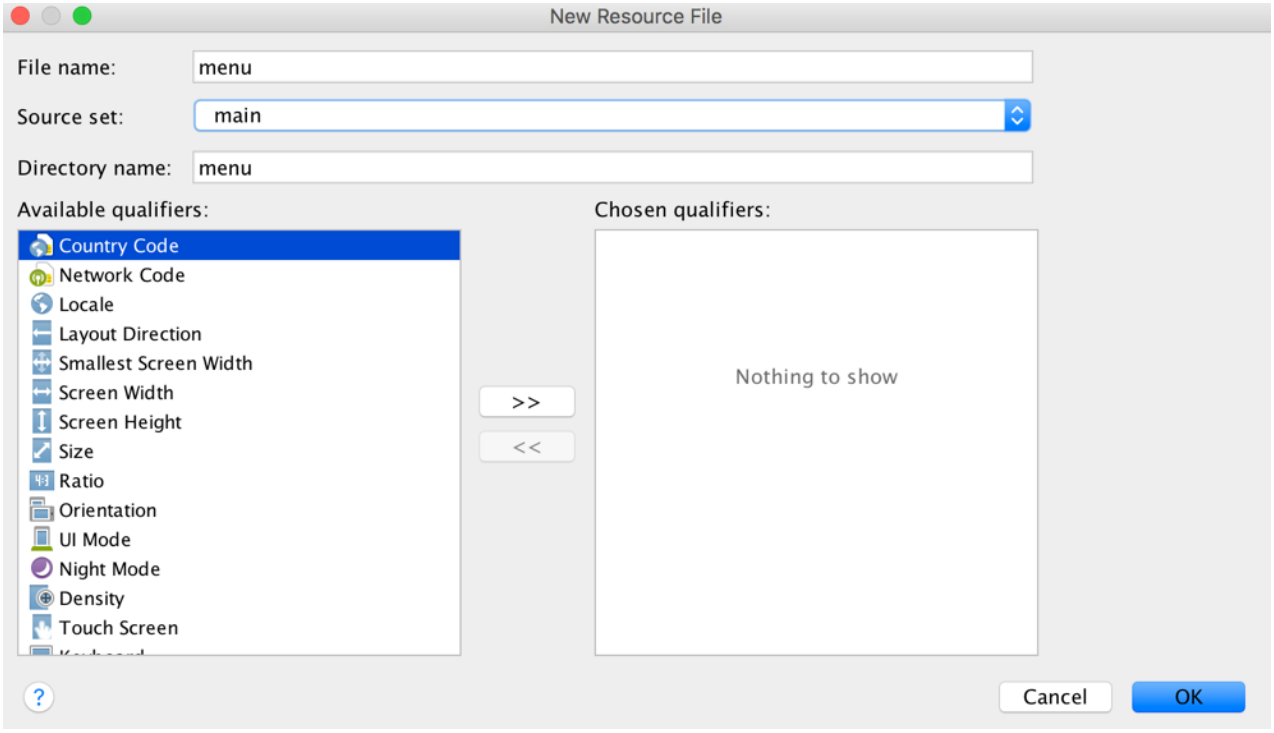



**Step 5:** Create new Resource directory named menu and new resource file named menu

Right click res directory -> New -> Android Resource Directory -> resource type -> select menu -> finish

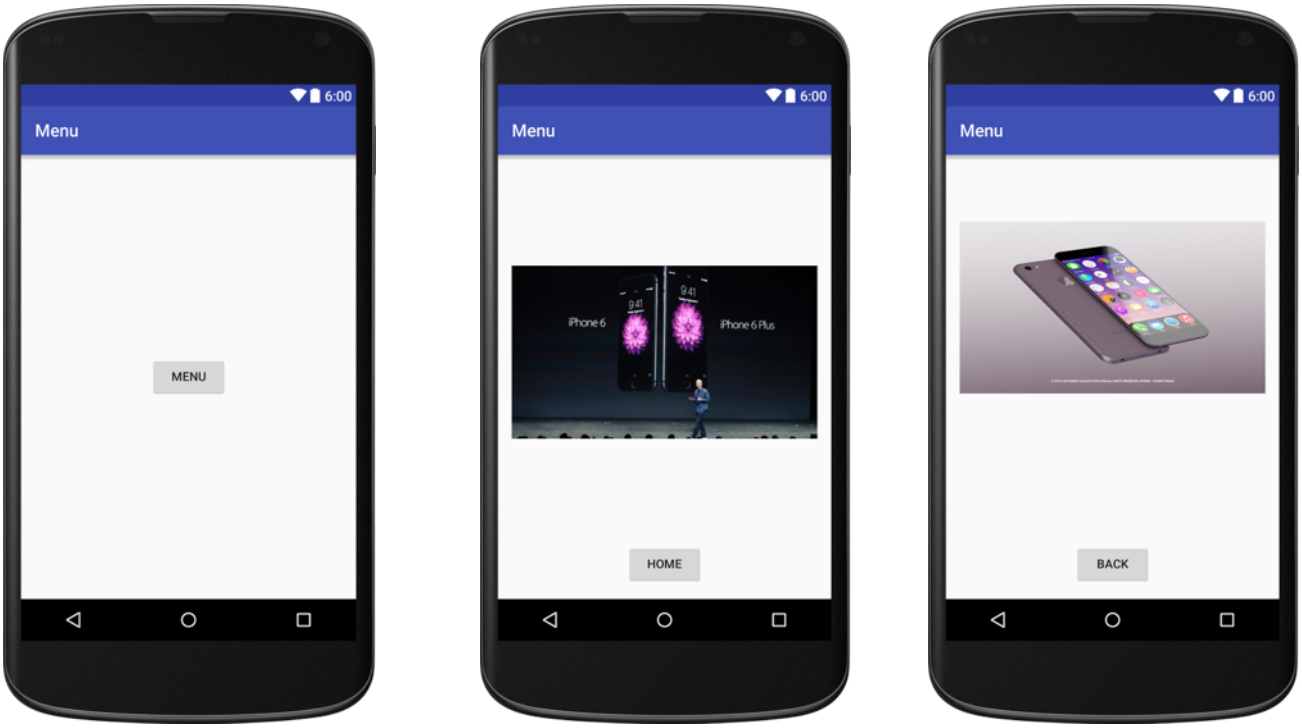


Right click menu directory -> New -> new menu resource file ->  enter file name -> Ok




**Step 4:** Design (After the design, the xml code will be generated automatically in the layout file )

**activity_main.xml** **activity_second.xml** **activity_third.xml**



**Step 5:** Open **menu.xml** and add the following code

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
   tools:context="com.example.kamarajios33.menu.MainActivity">
   <item
      android:id="@+id/one"
      android:title="One"/>
   <item
      android:id="@+id/two"
      android:title="Two"/>
</menu>
```

**Step 6:** Open **MainActivity.java**, **second.java & third.java** and add the following code

**MainActivity.java**

```
public class MainActivity extends AppCompatActivity {
   Button b1;


   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      b1=(Button)findViewById(R.id.b1);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            PopupMenu pm = new PopupMenu(MainActivity.this,b1);
            pm.getMenuInflater().inflate(R.menu.menu,pm.getMenu());
            pm.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
               @Override
               public boolean onMenuItemClick(MenuItem item) {
                  switch (item.getItemId()) {
                     case R.id.one:
                        Intent o = new Intent(getApplicationContext(), second.class);
                        startActivity(o);
                        System.exit(0);
                        break;
```

```
                    case R.id.two:
                        Intent in = new Intent(getApplicationContext(),third.class);
                        startActivity(in);
                        System.exit(0);
                        break;
                }

                return false;
            }
        });

        pm.show();
    }
  });
 }
}
```

**second.java**

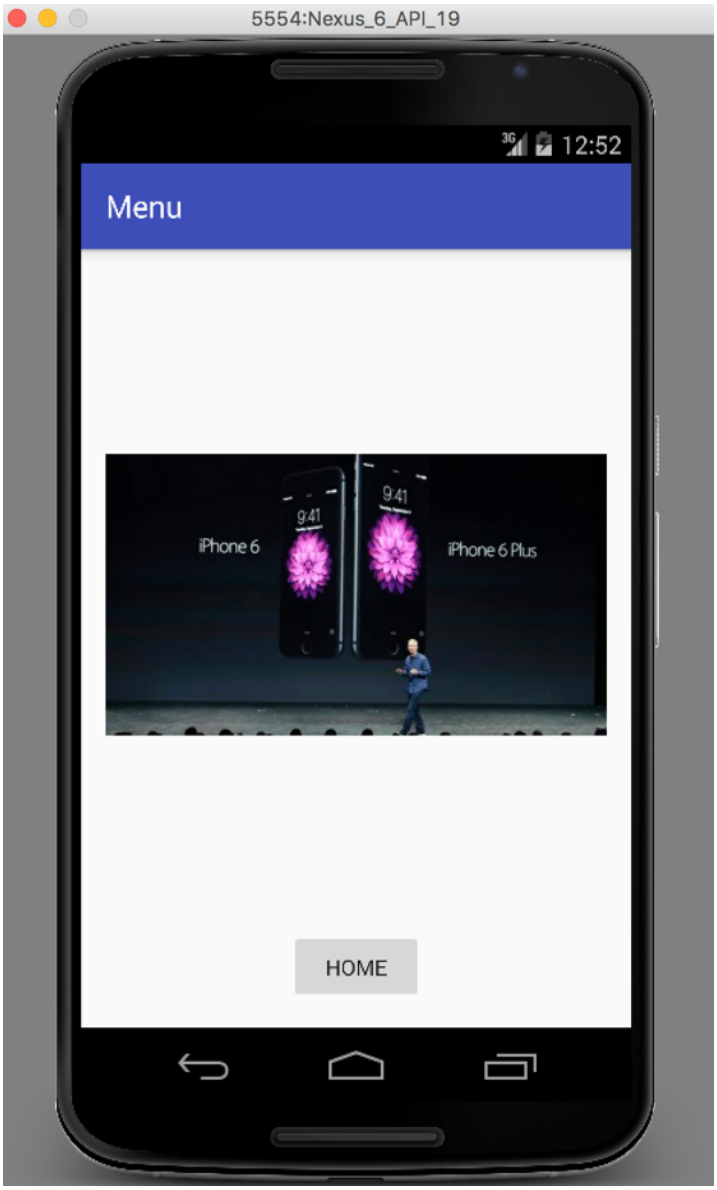
```
public class second extends AppCompatActivity {
   Button b2;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_second);
      b2=(Button)findViewById(R.id.b2);
      ImageView iv = (ImageView)findViewById(R.id.iw);
      b2.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent i = new Intent(getApplicationContext(),MainActivity.class);
            startActivity(i);
            System.exit(0);
         }
      });
   }
}
```

**Third.java**

```
public class third extends AppCompatActivity {
   Button back;
   ImageView img;




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

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_third);
        back=(Button)findViewById(R.id.back);
        img=(ImageView)findViewById(R.id.imageView);
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getApplicationContext(),MainActivity.class);
                startActivity(i);
                System.exit(0);
            }
        });
    }
}
```

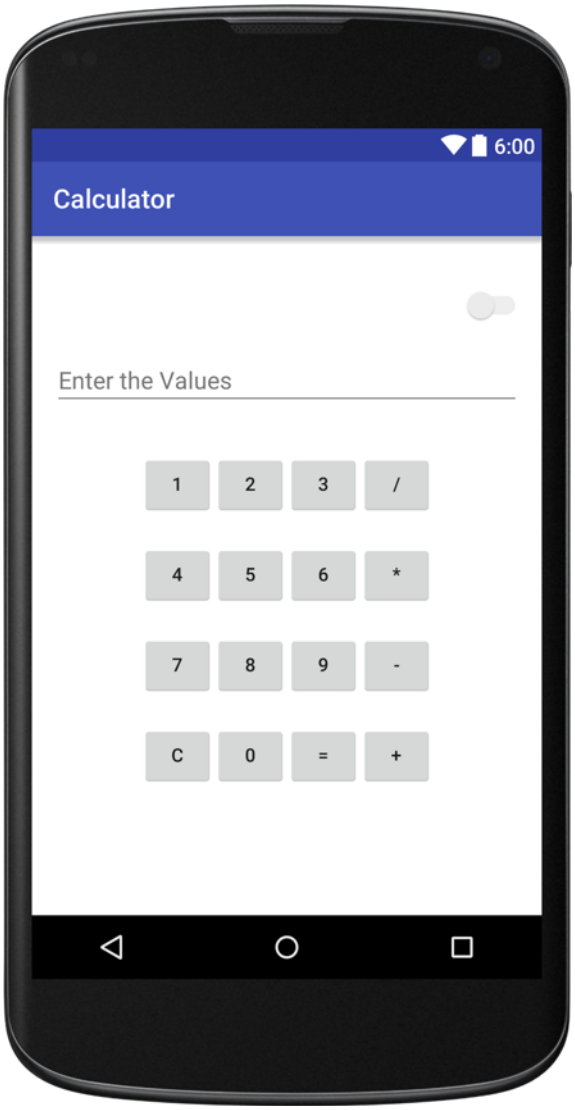
**Step 7:** The output of the above code is as follows.

# 4. Native Calculator Application

We create native calculator application in android. This application will perform mathematical operation into the android device. Let us design for UI in activity_main.xml

**Step 1:** Design for UI in layout file (activity_main.xml)



**activity_main.xml**

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".Home" android:background="#fff"
   android:weightSum="1">

   <LinearLayout
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="72dp"
      android:id="@+id/onoff">

      <Switch
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         android:id="@+id/switch1"
         android:checked="false" />
   </LinearLayout>

   <LinearLayout
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="55dp"
      android:layout_gravity="center_horizontal"
      android:id="@+id/l1">
```

```
<EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id = "@+id/display"
        android:hint="Enter the Values"
        android:layout_weight="1" />
  </LinearLayout>

  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:paddingTop="20dp"
    android:id="@+id/l5">
    <Button android:layout_width="55dp"
      android:layout_height="wrap_content"
      android:id = "@+id/one"
      android:text="1" />

    <Button android:layout_width="55dp"
      android:layout_height="wrap_content"
      android:id = "@+id/two"
      android:text="2" />
    <Button android:layout_width="55dp"
      android:layout_height="wrap_content"
      android:id = "@+id/three"
      android:text="3" />
    <Button android:layout_width="55dp"
      android:layout_height="wrap_content"
      android:id = "@+id/div"
      android:text="/" />

  </LinearLayout>
```

```
<LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:gravity="center"
        android:paddingTop="20dp"
        android:id="@+id/l2">
        <Button
           android:layout_width="55dp"
           android:layout_height="wrap_content"
           android:id = "@+id/four"
           android:text="4" />
        <Button
           android:layout_width="55dp"
           android:layout_height="wrap_content"
           android:id = "@+id/five"
           android:text="5" />
        <Button
           android:layout_width="55dp"
           android:layout_height="wrap_content"
           android:id = "@+id/six"
           android:text="6" />
        <Button
           android:layout_width="55dp"
           android:layout_height="wrap_content"
           android:id = "@+id/mul"
           android:text="*" />
    </LinearLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:paddingTop="20dp"
        android:id="@+id/l3">
```

```
    <Button android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/seven"
        android:text="7" />
    <Button android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/eight"
        android:text="8" />
    <Button android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/nine"
        android:text="9" />
    <Button android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/sub"
        android:text="-" />
</LinearLayout>
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:paddingTop="20dp"
    android:id="@+id/l4">
    <Button
        android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/cancel"
        android:text="C" />
    <Button
        android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/zero"
        android:text="0" />
    <Button
        android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/equal"
        android:text="=" />
```

```
<Button
        android:layout_width="55dp"
        android:layout_height="wrap_content"
        android:id = "@+id/add"
        android:text="+" />
</LinearLayout>
</LinearLayout>
```

**Step 2:** Open MainActivity.java and add following code

```
package com.example.kamarajios33.calc;

import android.os.Bundle;
import android.app.Activity;
import android.support.annotation.RequiresPermission;
import android.text.Editable;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Switch;

public class MainActivity extends Activity implements View.OnClickListener
{
   Button one, two, three, four, five, six, seven, eight, nine, zero, add, sub, mul,
div, cancel, equal;
   EditText disp;
   int op1;
   int op2;
   String optr;
   Switch onoff;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      onoff = (Switch)findViewById(R.id.switch1);
      onoff.setChecked(true);
      one = (Button) findViewById(R.id.one);
      two = (Button) findViewById(R.id.two);
```

```
        three = (Button) findViewById(R.id.three);
        four = (Button) findViewById(R.id.four);
        five = (Button) findViewById(R.id.five);
        six = (Button) findViewById(R.id.six);
        seven = (Button) findViewById(R.id.seven);
        eight = (Button) findViewById(R.id.eight);
        nine = (Button) findViewById(R.id.nine);
        zero = (Button) findViewById(R.id.zero);
        add = (Button) findViewById(R.id.add);
        sub = (Button) findViewById(R.id.sub);
        mul = (Button) findViewById(R.id.mul);
        div = (Button) findViewById(R.id.div);
        cancel = (Button) findViewById(R.id.cancel);
        equal = (Button) findViewById(R.id.equal);
        disp = (EditText) findViewById(R.id.display);
        onoff.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
                if(isChecked)
                {

                    one.setEnabled(true);
                    two.setEnabled(true);
                    three.setEnabled(true);
                    four.setEnabled(true);
                    five.setEnabled(true);
                    six.setEnabled(true);
                    seven.setEnabled(true);
                    eight.setEnabled(true);
                    nine.setEnabled(true);
                    zero.setEnabled(true);
                    add.setEnabled(true);
                    sub.setEnabled(true);
                    mul.setEnabled(true);
                    sub.setEnabled(true);
```

```
            mul.setEnabled(true);
            div.setEnabled(true);
            cancel.setEnabled(true);
            equal.setEnabled(true);
            disp.setEnabled(true);

        }
        else
        {
            one.setEnabled(false);
            two.setEnabled(false);
            three.setEnabled(false);
            four.setEnabled(false);
            five.setEnabled(false);
            six.setEnabled(false);
            seven.setEnabled(false);
            eight.setEnabled(false);
            nine.setEnabled(false);
            zero.setEnabled(false);
            add.setEnabled(false);
            sub.setEnabled(false);
            mul.setEnabled(false);
            sub.setEnabled(false);
            mul.setEnabled(false);
            div.setEnabled(false);
            cancel.setEnabled(false);
            equal.setEnabled(false);
            disp.setEnabled(false);
        }
    }
});
try {
    one.setOnClickListener(this);
    two.setOnClickListener(this);
    three.setOnClickListener(this);
    four.setOnClickListener(this);
    five.setOnClickListener(this);
```

```
        six.setOnClickListener(this);
        seven.setOnClickListener(this);
        eight.setOnClickListener(this);
        nine.setOnClickListener(this);
        zero.setOnClickListener(this);
        cancel.setOnClickListener(this);
        add.setOnClickListener(this);
        sub.setOnClickListener(this);
        mul.setOnClickListener(this);
        div.setOnClickListener(this);
        equal.setOnClickListener(this);
    } catch (Exception e) {
    }
}

public void operation() {
    if (optr.equals("+")) {
        op2 = Integer.parseInt(disp.getText().toString());
        disp.setText("");
        op1 = op1 + op2;
        disp.setText(Integer.toString(op1));
    } else if (optr.equals("-")) {
        op2 = Integer.parseInt(disp.getText().toString());
        disp.setText("");
        op1 = op1 - op2;
        disp.setText(Integer.toString(op1));
    } else if (optr.equals("*")) {
        op2 = Integer.parseInt(disp.getText().toString());
        disp.setText("");
        op1 = op1 * op2;
        disp.setText(Integer.toString(op1));
    } else if (optr.equals("/")) {
        op2 = Integer.parseInt(disp.getText().toString());
        disp.setText("");
        op1 = op1 / op2;
        disp.setText(Integer.toString(op1));
    }
}
```

```
@Override
public void onClick(View arg0)
{
   Editable str = disp.getText();
   switch (arg0.getId()) {
      case R.id.one:
         if (op2 != 0) {
            op2 = 0;
            disp.setText("");
         }
         str = str.append(two.getText());
         disp.setText(str);
         break;
      case R.id.two:
         if (op2 != 0) {
            op2 = 0;
            disp.setText("");
         }
         str = str.append(two.getText());
         disp.setText(str);
         break;
      case R.id.three:
         if (op2 != 0) {
            op2 = 0;
            disp.setText("");
         }
         str = str.append(three.getText());
         disp.setText(str);
         break;
      case R.id.four:
         if (op2 != 0) {
            op2 = 0;
            disp.setText("");
         }
         str = str.append(four.getText());
         disp.setText(str);
         break;
```

```
case R.id.five:
    if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    str = str.append(five.getText());
    disp.setText(str);
    break;
case R.id.six:
    if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    str = str.append(six.getText());
    disp.setText(str);
    break;
case R.id.seven:
    if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    str = str.append(seven.getText());
    disp.setText(str);
    break;
case R.id.eight:
    if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    str = str.append(eight.getText());
    disp.setText(str);
    break;
```

```
case R.id.nine:
    if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    str = str.append(nine.getText());
    disp.setText(str);
    break;
case R.id.zero:
    if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    str = str.append(zero.getText());
    disp.setText(str);
case R.id.cancel:
    op1 = 0;
    op2 = 0;
    disp.setText("");
    disp.setHint("Perform Operation");
    break;
case R.id.add:
    optr = "+";
    if (op1 == 0) {
        op1 = Integer.parseInt(disp.getText().toString());
        disp.setText("");
    }
    else if (op2 != 0) {
        op2 = 0;
        disp.setText("");
    }
    else
    {
        op2 = Integer.parseInt(disp.getText().toString());
        disp.setText("");
        op1 = op1 + op2;
        disp.setText(Integer.toString(op1));
    }
    break;
```

```
case R.id.sub:
    optr = "-";
    if (op1 == 0)
    {
       op1 = Integer.parseInt(disp.getText().toString());
       disp.setText("");
    }
    else if (op2 != 0) {
       op2 = 0;
       disp.setText("");
    }
    else
    {
       op2 = Integer.parseInt(disp.getText().toString());
       disp.setText("");
       op1 = op1 - op2;
       disp.setText(Integer.toString(op1));
    }
    break;
 case R.id.mul:
    optr = "*";
    if (op1 == 0)
    {
       op1 = Integer.parseInt(disp.getText().toString());
       disp.setText("");
    }
    else if (op2 != 0) {
       op2 = 0;
       disp.setText("");
    }
    else
    {
       op2 = Integer.parseInt(disp.getText().toString());
       disp.setText("");
       op1 = op1 * op2;
       disp.setText(Integer.toString(op1));
    }
    break;
```

```
  case R.id.div:
    optr = "/";
    if (op1 == 0)
    {
      op1 = Integer.parseInt(disp.getText().toString());
       disp.setText("");
    }
    else if (op2 != 0) {
       op2 = 0;
       disp.setText("");
    }
    else
    {
       op2 = Integer.parseInt(disp.getText().toString());
       disp.setText("");
       op1 = op1 / op2;
       disp.setText(Integer.toString(op1));
    }
    break;
case R.id.equal:
    if (!optr.equals(null))
    {
       if (op2 != 0)
       {
          if (optr.equals("+"))
          {
             disp.setText("");
             op1 = op1 + op2;
             disp.setText(Integer.toString(op1));
          }
          else if (optr.equals("-"))
          {
             disp.setText("");
             op1 = op1 - op2;
             disp.setText(Integer.toString(op1));
          }
```

```
        else if (optr.equals("*"))
         {
            disp.setText("");
            op1 = op1 * op2;
            disp.setText(Integer.toString(op1));
         }
        else if (optr.equals("/"))
         {
            disp.setText("");
            op1 = op1 / op2;
            disp.setText(Integer.toString(op1));
         }
      }
      else
      {
         operation();
      }

   }
   break; }}}
```

**Step 3:** The output of the above code is as follows.


# 5. Graphical Primitives

**Step1:** Create an android project using android studio.

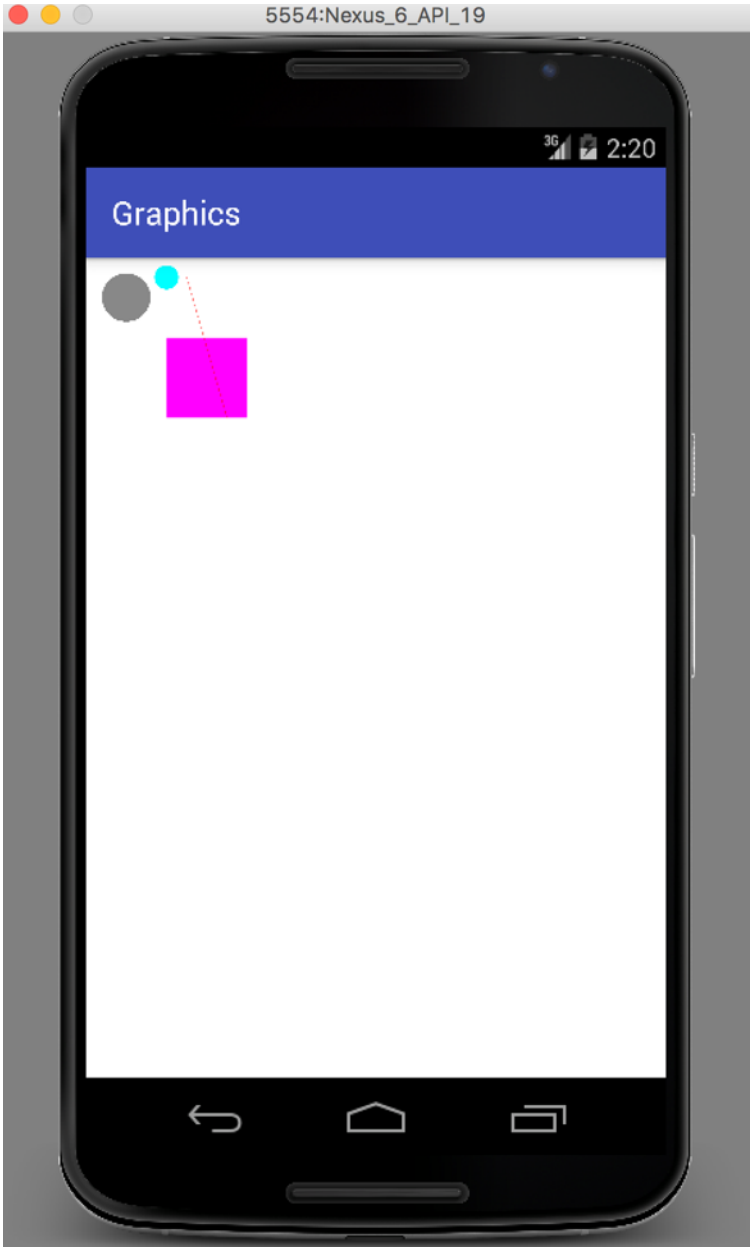
**Step 2:** After creating the project, open the java file named MainActivity.xml.

**MainActivity.java**

```
package com.example.kamarajios33.graphics;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.RectF;
import android.view.View;

public class MainActivity extends AppCompatActivity {
   DemoView dv;

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

      dv = new DemoView(this);
      setContentView(dv);
   }

   private class DemoView extends View
   {
      public DemoView(Context context)
      {
         super(context);
      }
      @Override
      protected void onDraw(Canvas canvas)
      {
         super.onDraw(canvas);
         Paint ob=new Paint();
         ob.setStyle(Paint.Style.FILL);
```

```
ob.setColor(Color.WHITE);
canvas.drawPaint(ob);
ob.setColor(Color.GRAY);
canvas.drawCircle(100, 100, 60, ob);
ob.setColor(Color.CYAN);
canvas.drawCircle(200, 50, 30, ob);
ob.setColor(Color.MAGENTA);
canvas.drawRect(200, 200, 400, 400, ob);
ob.setColor(Color.RED);
canvas.drawLine(250,50,350,400,ob);
canvas.rotate(-45);        }}}
```

**Step 3:** The output of the above code is as follows.



# 6. Persistent Data Storage

## Objectives

After completing this chapter we will be able to understand the following topics.

- Introduction to SQLite
- Android Database API
- Using Content Provider
- Connecting with Database Server

## 6.1 Introduction to SQLite

Android SQLite database is an integral part "built-in" component. SQLite is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. It is a self-contained, transactional database engine that requires no separate server process.To use this database the knowledge of SQL alone sufficient. We need not to learn any new query language.

SQLite supports the data types such as TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java). All other types must be converted into one of these fields before getting saved in the database. SQLite does not validate,whether the data stored in to the columns are actually of the defined type, e.g. wecan write an integer into a string column and vice versa.

SQLite is embedded into every android device. Since the database requires limited memory at runtime (approx. 250 KByte) it is widely chosen for embedding in devices. Android platform includes the SQLite embedded database and provides APIs to access the database. To use the SQLite database in android,we donot require a setup procedure.

We need to only have to define the SQL statements for creating and updating the database. Thenthe database will be managed by the android system. Access to the SQLite database involves accessing the file system. This can be slow. Therefore it is recommended to perform database operations asynchronously. If our application creates a database, this database is stored by default in the following directory DATA/data/APP_NAME/databases/FILENAME.

The parts of the above directory have been explained below..

- DATA is the path which the Environment.getDataDirectory() method returns.
- APP_NAME is our application name.
- FILENAME is the name wespecify in our code for the database.

## 6.2 Android Database API

The android.database package contains all necessary classes for working with databases. The android.database.sqlite package contains the SQLite specific classes. The following classes will be helpful while developing app that uses database.

- SQLiteDataBase
- Cursor
- SQLiteOpenHelper

1. **SQLiteDatabase**

SQLiteDatabase is the base class for working with a SQLite database in android and provides methods to open, query, update and close the database. In addition it provides the execSQL() method, which allows to execute an SQL statement directly.

To perform the following database operations, SQLiteDatabase class provides various methods.

a) **Insert operation** – For inserting values to the database, we need to save data into contentvalues. ContentValues allow defining key/value pairs. The key represents the table column name and the value represents the content corresponding to that column.

The object ContentValues allows to define key/values. The key represents the table column identifier and the value represents the content for the table record in this column. ContentValues can be used for inserts and updates of database entries.

```
SQLiteDatabase db = this.getWritableDatabase();
ContentValues values = new ContentValues();
        values.put(column1, value1);
        values.put(column2, value2);
        values.put(column3, value3);
        values.put(column4, value4);
// insert row in table
```

b) **Update Operation:** To update the existing record in your Android sqlite database table, you need to execute the following method.

```
db.update(TABLE_NAME,content_values,WHERE_CLAUSE,WHERE_ARGUMENTS);
```

- The first argument takes the name of the table.
- The second argument takes a new record. Here, we pass the contentvalue object which contains data corresponding to each column in a key/value pair. So, all the columns which need to get updated are provided in the contentvalue object as a key/value pair.

- The third argument specifies the where clause (it's a condition). For example, update the employee name where emloyee ID is "123". Here, the where clause is the "id=".
- The fourth argument takes the value corresponding to the where clause. For the above example it is "123".

c) **Delete Operation**: Similarly, you can perform the delete operation on an Android SQLite database table. It takes three arguments. The first is the table name, the second is the where clause, and the third is the value corresponding to that where clause.

```
db.delete(TABLE_NAME,WHERE_CLAUSE,WHERE_ARGUMENTS);
```

d) **Read Operation**: To read values from an Android SQLite database table, we need to learn about cursors. We execute the select operation on the database and we get multiple rows as a result. We assign those rows to a cursor. A Cursor points to one row at a time. This is how we get the required data.

Raw Query: This query directly accepts SQL as an input. You can pass SQL statement(s) and SQLiteDatabase will execute that query for you.

```
db.rawQuery(sql_statement,null);
```

2. **Cursor**

A query returns a Cursor object. A Cursor represents the result of a query and basically points to one row of the query result. This way Android can buffer the query results efficiently; as it does not have to load all data into memory.

To get the number of elements of the resulting query use the getCount() method. To move between individual data rows, you can use the moveToFirst() and moveToNext() methods. The isAfterLast() method allows to check if the end of the query result has been reached.

Cursor provides typed get*() methods, e.g. getLong(columnIndex), getString(columnIndex) to access the column data for the current position of the result. The "columnIndex" is the number of the column you are accessing. Cursor also provides the getColumnIndexOrThrow(String) method which allows to get the column index for a column name of the table. A Cursor needs to be closed with the close() method call.

In the following code, we assign the result of the select statement to the cursor and manipulate the data.

```
String selectQuery  =  "SELECT              * FROM TABLE_NAME";
Cursor c = db.rawQuery(selectQuery, null);

//check if query result is not null. if (c != null)
c.moveToFirst();

String col1 = c.getString(c.getColumnIndex(column_name)); int col2 =
c.getInt(c.getColumnIndex(column_name));
```

## SQLiteOpenHelper

To create and upgrade a database in our Android application we can usef the SQLiteOpenHelper class.

- onCreate() - is called by the framework, if the database is accessed but not yet created.
- onUpgrade() - called, if the database version is increased in your application code. This method allows you to update an existing database schema or to drop the existing database and recreate it via the onCreate() method.
- Both methods receive an SQLiteDatabase object as parameter which is the Java representation of the database.
- The SQLiteOpenHelper class provides the getReadableDatabase() and getWriteableDatabase() methods to get access to an SQLiteDatabase object; either in read or write mode.

The database tables should use the identifier _id for the primary key of the table. Several Android functions rely on this standard.

**Enhancing the display of Records**

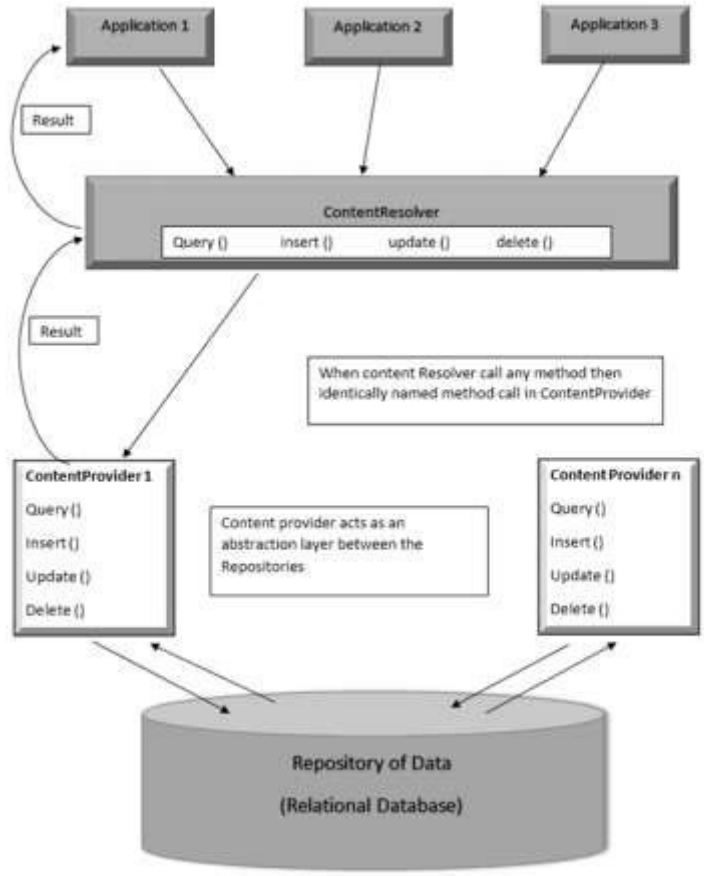
After retrieving the records, we need to display them to user. While displaying records, we need to choose the UI control. The following UI controls will be helpful for displaying more number of records from database to the user.

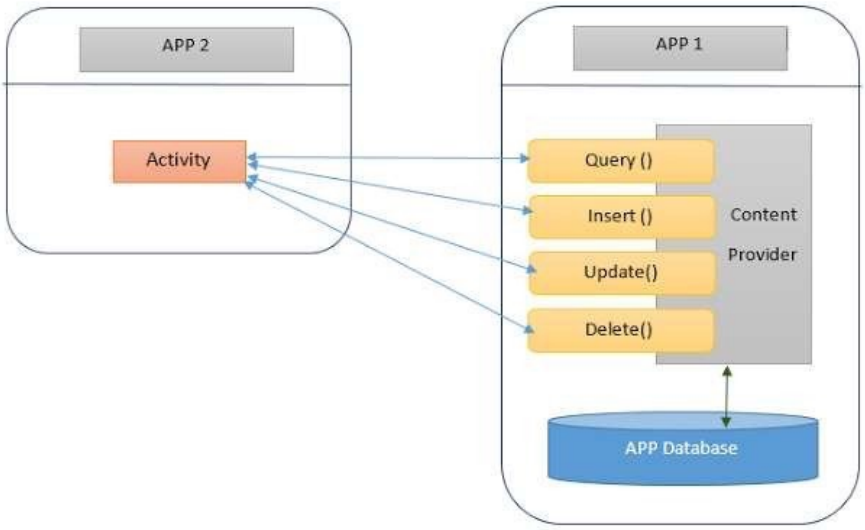
- ListViews are Views which allow to display a list of elements.
- ListActivities are specialized activities which make the usage of ListViews easier.
- The SimpleCursorAdapter class will map the columns to the Views based on the Cursor passed to it. The SimpleCursorAdapter allows to set a layout for each row of the ListViews.

You also define an array which contains the column names and another array which contains the IDs of Views which should be filled with the data

## Using Content Provider

The security model of android does not allow an application to access data from another application. Still we may want to share data from our app to another app or our app might need data which can be accessed from another app. Content provider is the best way to share data across applications. Content provider is a set of data wrapped up in a custom API to read and




write. Applications/Processes have to register themselves as a provider of data. Other applications can request android system to read/write that data through a predefinedAPI. Content provider API adheres to CRUD principle.

**Content Provider Workflow :**

Examples for content provider are Contactsthat exposes user information to other applications, Media store which allows other applications to access and store media files etc.

Content providers are simple interfaces which uses standard insert(), query(), update(), delete() methods to access app data. A special URI starting with content:// will be assigned to each content providers and that will be recognized across applicationsor apps.

Above diagram shows how content provider works. App 1 stores its data in its own database and provides a content provider. App 2 communicates access App 1's data through the content provider.

**Writing a Content Provider**

To the following are the steps for creating a content provider.

1) Create sub class for ContentProvider.
2) Define content URI
3) Implement or override the required methods. insert(), update(), query(), delete(), getType().
4) Declare the content provider in AndroidManifest.xml

The steps are explained below with description.

1) **Create a subclass for ContentProvider class.**
   Create a class and extend the ContentProvider class available in the android.content package.

2) **Defining URI:**
   Content URI has a special path similar to HTTP. Content provider URI consists of four parts.

   content://authority/path/id

   - content:// All the content provider URIs should start with this value
   - 'authority' is Java namespace of the content provider implementation. (fully qualified Java package name)
   - 'path' is the virtual directory within the provider that identifies the kind of data being requested.
   - 'id' is optional part that specifies the primary key of a record being requested. We can omit this part to request all records.

3) **Overriding the required methods**
   a) Adding new records:
   We need to override insert() method of the ContentProvider to insert new record to the database via content provider. The caller method will have to specify the content provider URI and the values to be inserted without the ID. Successful insert operation will return the URI value with the newly inserted ID.

   b) Updating records
   To update one or more records via content provider we need to specify the content provider URI. update() method of the ContentProvider is used to update records. We have to specify the ID of the record to be updated. To update multiple records, we have to specify 'selection' parameter to indicate which rows are to be updated. This method will return the number of rows updated.

c) Deleting records

Deleting one or more records is similar to update operation. We need to specify either ID or 'selection' to delete records. The delete() method of the ContentProvider will return the number of records deleted.

d) Querying the content provider

To query the data via content provider we overridequery() method of ContentProvider. This method has many parameters. We can specify list of columns that will be placed in the result cursor using 'projection' parameter. We can specify 'selection' criteria, 'sortOrder' etc. If we do not specify ‘projection’ parameter, all the columns will be included in the result cursor. If we do not specify sortOrder the provider will choose its own sort order.

e) getType() method

This method is used to handle requests for the MIME type of data at the given URI. We use either vnd.android.cursor.item or vnd.android.cursor.dir/ The vnd.android.cursor.item is used to represent specific item. Another one is used to specify all items.

4) **Registering the provider in AndroidManifest.xml**

Content providers need to be registered in the AndroidManifest.xml. To register the content providers in manifest file, the <provider> element is used.The <application> element is the parent tag of the <provider> element.

```
<provider
android:name=".MyProvider"
android:authorities="com.example.contentproviderexample.MyProvider">
```

Here authorities is the URI authority to access the content provider. Typically this will be the fully qualified name of the content provider class.

## 6.3 Running the SQLite App

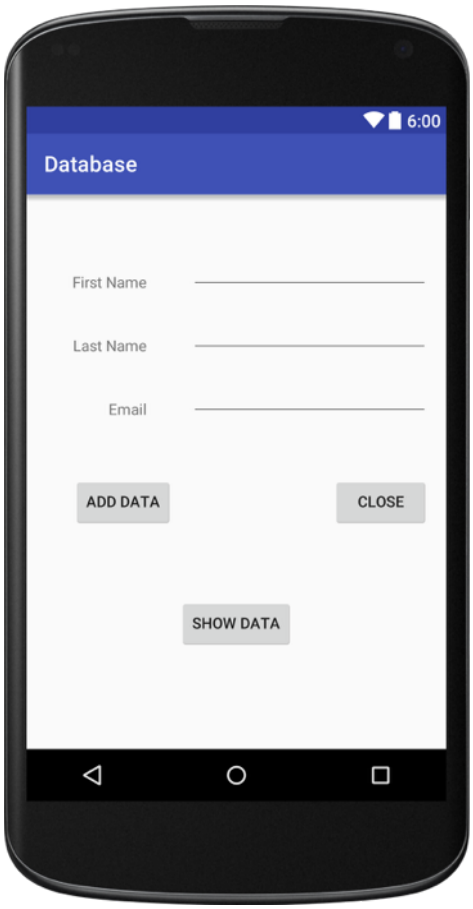
**Step 1:** Create an android project using android studio

**Step 2:** Create two resource files (*.xml) and two activity files (*.java) named activity_main.xml & activity_main2.xml and MainActivity.java & Main2Activity.java.

**Step 3:** Open res directory -> layout -> activity_main.xml -> Click -> Design button at bottom of the Android Studio. Put the necessary components for both resource files (activity_main.xml, activity_main2.xml)

**Step 4:** Design (After the design part, the xml code will be generated automatically in the layout file )

activity_main.xml




activity_main2.xml


**Step 5:** Create an activity file

**MainActivity.java**

```
package com.example.kamarajios33.database;

import android.app.AlertDialog;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TableRow;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   SQLiteDatabase db;
   String f,l,e;
   @Override
   protected void onCreate(Bundle savedInstanceState)

   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      db = openOrCreateDatabase("Mydb",MODE_PRIVATE,null);
            db.execSQL("CREATE TABLE IF NOT EXISTS student(fname VARCHAR,lname
VARCHAR,email VARCHAR);");

   }

   public void Adddata(View view)
   {
      EditText e1 = (EditText)findViewById(R.id.e1);
      EditText e2 = (EditText)findViewById(R.id.e2);
      EditText e3 = (EditText)findViewById(R.id.e3);
      f = e1.getText().toString();
      l=e2.getText().toString();
      e=e3.getText().toString();
      db.execSQL("INSERT INTO student VALUES('"+f+"','"+l+"','"+e+"');");
                        Toast.makeText(getApplicationContext(),"Row
Inserted",Toast.LENGTH_SHORT).show();
   }
```

```
public void close(View view)
    {
        System.exit(0);
    }
    public void showdata(View view)
    {
        //Toast.makeText(getApplicationContext(),"Show Data",Toast.LENGTH_LONG).show();
        Intent i = new Intent(getApplicationContext(),Main2Activity.class);
        startActivity(i);
    }
}
```

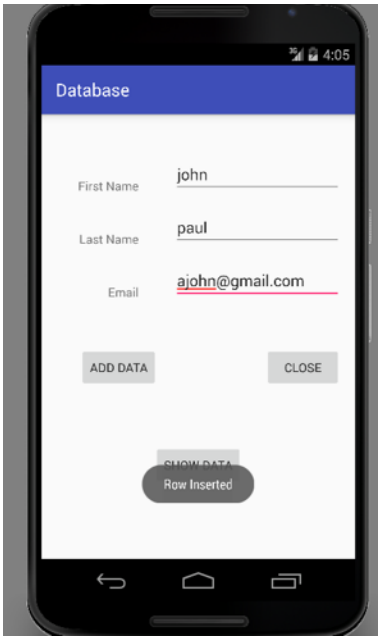
**Main2Activity.java**

```
package com.example.kamarajios33.database;

import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Main2Activity extends AppCompatActivity {
    EditText fn,ln,em;
    Button next,back,prev;
    SQLiteDatabase db;
    Integer j =0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.activity_main2);
fn=(EditText)findViewById(R.id.fn);
ln=(EditText)findViewById(R.id.ln);
em=(EditText)findViewById(R.id.em);
back =(Button)findViewById(R.id.button2);
next=(Button)findViewById(R.id.button);
prev=(Button)findViewById(R.id.b3);

db = openOrCreateDatabase("Mydb",MODE_PRIVATE,null);

final Cursor c = db.rawQuery("select * from student",null);
c.moveToFirst();
fn.setText(c.getString(c.getColumnIndex("fname")));
ln.setText(c.getString(c.getColumnIndex("lname")));
em.setText(c.getString(c.getColumnIndex("email")));

back.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      Intent il = new Intent(getApplicationContext(), MainActivity.class);
      startActivity(il);
      System.exit(0);
   }
});
next.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      try
      {
         c.moveToNext();
         fn.setText(c.getString(c.getColumnIndex("fname")));
         ln.setText(c.getString(c.getColumnIndex("lname")));
         em.setText(c.getString(c.getColumnIndex("email")));
      }
```

```
catch (Exception e)
            {
    Toast.makeText(getApplicationContext(),"Last Record",Toast.LENGTH_LONG).show();
                e.printStackTrace();
            }
        }
    });
    prev.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try
            {
                c.moveToPrevious();
                fn.setText(c.getString(c.getColumnIndex("fname")));
                ln.setText(c.getString(c.getColumnIndex("lname")));
                em.setText(c.getString(c.getColumnIndex("email")));
            }
            catch (Exception e)
            {
    Toast.makeText(getApplicationContext(),"First Record",Toast.LENGTH_LONG).show();
                e.printStackTrace();
            }}
    });
  }}
```

**Step 6:** Run the project. While running, the following output will be shown in the emulator.




## WebView

Android WebView component used in Android web app to render the android apps GUI. Webview component is a browser supported view and it allows us implementing web pages in our android application as we like. WebView gives most of the screen space in our android application.

- Android Web App or Android Hybrid App

  Android web app is actually a mix of web app and native android app. The app is developed using support of Android components and web technologies inside a WebView. It contain web technologies like HTML, CSS, JavaScript, SVG, Json, Ruby, Rails etc., Another common term for an Android web app is Android hybrid app.
- WebView is Based on Chrome

  The WebView component is fully based on same code as Chrome for android. It indicates that web app which supported in Chrome browser that can be converted into Android App. From Android 4.4 Chrome replacing the old android browser as default/ builtin browser.
- WebView Needs Internet Permission

  In the Android application Internet permission is mandatory if app needs to load a webpage. While installing an app the user should accepts the internet access permission.To get the internet permission in the android app, internet permission element is to add in manifest file.

Example of an Android manifest file with internet permission:

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


<uses-permission android:name="android.permission.INTERNET" />

<application ...>
</application>

</manifest>
```

It is the XML element <uses-permission android:name="android.permission.INTERNET" /> (via the android:name attribute) which signals that the app needs internet permission.

**Using WebView**

In order to use the Android WebView component We need to insert that component in our layout. This is most often done by inserting a WebView element into the layout XML file for the layout we want the WebView to be displayed in. Here is an example layout file with a WebView embedded:

```
<RelativeLayoutxmlns:android="http://schemas.android.com/apk/res/and roid"

xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent" android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

<WebView android:id="@+id/webview"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"></WebView>

</RelativeLayout>
```

Once we have inserted a WebView into a layout somewhere, we can access it from our code. We need to access the WebView to make it do any interesting task for user. Typically WebView control is accessed from inside an activity. Here is the sample code which accesses a WebView embedded in its layout XML file:

```
public class MainActivity extends Activity {

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

WebViewwebView = (WebView)
findViewById(R.id.webview)
WebViewwebView = (WebView) findViewById(R.id.webview); webView.loadUrl("http://

www.kcetvnr.orgn");

      }
}
```

Once inserted WebView control in the layout, we can refer it inside the activity class and can instruct it to load URLs via HTTP or any other task. Here we have used the WebView's loadUrl() method which loads the URL into the WebView.

# 7. Multithreading

**Objectives**

After completing this chapter we will be able to understand the following topics.

- Multithreading
- Multimedia in Android
- Media Player API
- Camera API
- Running the Multithreading App

**Multithreading**

Multi-threading is defined as a feature through which we can run two or more concurrent threads of a process. In this a process, the common data is shared among all these threads also known as sub-processes exclusively.

**Multimedia in Android**

Multimedia capabilities, or the playing and recording of audio and video, is one such significant task that many users find to be of great value. Take a quick look around us, we will find people using the phone as a means to enjoy a variety of programs as well as share self-recorded media among friends. Android provides APIs to easily access this capability as well as embed multimedia and its manipulation directly within an application. Usually multimedia involves in performing the following basic functions.

- Playing audio and video
- Controlling the camera
- Recording audio
- Recording video

We will discuss the 'Stagefright' architecture from Google. Because the foundation of Android's multimedia platform is Google's new media platform Stagefright. Stagefright, as of now, supports the following core media files, services, and features:

- Interfaces for third-party and hardware media codecs, input and output devices, and content policies.

- Media playback, streaming, downloading, and progressive playback, including third-Generation Partnership Program (3GPP), Moving Picture Experts Group 4 (MPEG-4), Advanced Audio Coding (AAC), and Moving Picture Experts Group (MPEG) Audio Layer 3 (MP3) containers.
- Network protocols including RTSP (TRP, SDP), HTTP progressive streaming, and HTTP live streaming.
- Video and image encoders and decoders, including MPEG-4, International Tele-communication Union H.263 video standard (H.263), Advanced Video Coding (AVC H.264), and the Joint Photographic Experts Group (JPEG).
- Speech codecs, including Adaptive Multi-Rate audio codecs AMR-NB and AMR-WB.
- Audio codecs, including MP3, AAC, and AAC+ etc.
- Media recording, including 3GPP, VP8, MPEG-4, and JPEG.
- Video telephony based on the 3GPP video conferencing standard 324-M.

## 7.1 Media Player API

**Android Multimedia Framework – android.media API**

The android multimedia framework provides developers a way to easily integrate audio and video playback into applications, and supports most of the common media types. The MediaPlayer class is the key in android multimedia framework. It can be used to play media on the local file system, media files stored in the application's resources, as well as data streaming over a network connection.

**Playing Audio**

Probably the most basic need for multimedia on a cell phone is the ability to play audio files, whether new ringtones, MP3s, or quick audio notes. Media- Player of android is easy to use. To play an MP3 file follow these steps:

- Place the MP3 in the res/raw directory in a project (note that we can also use a URI to access files on the network or via the internet).
- Create a new instance of the MediaPlayer, and reference the MP3 by calling MediaPlayer.create().
- Call the MediaPlayer methods prepare() and start().

**Playing Video**

Playing a video is slightly more complicated than playing audio with the MediaPlayer API, because we have to provide a view surface for our video to play on. Android has a VideoView widget that handles this task for us. This widget can be used with any layout manager. Android also provides a number of display options, including scaling and tinting.

## 7.2 Android Camera API

Usually every android device has at least one camera (back camera) or many devices have multiple cameras like front camera and rear camera. There is a default application available in each android device to access this camera. We can also access this application into our application as intent. Alternatively, we can also have our own camera application using android camera API. There is set of methods and class that support camera in android device.

**Camera API in Detail:**

We create intent to access camera in android. This intent can created using default camera application or using android camera API. we will create an application using camera API and will create intent to access this. This application will capture an image and will save it to the image directory into the android device. Once an image has been captured, this application will preview this image, while saving. Let us create a new java class that will capture an image as listed.

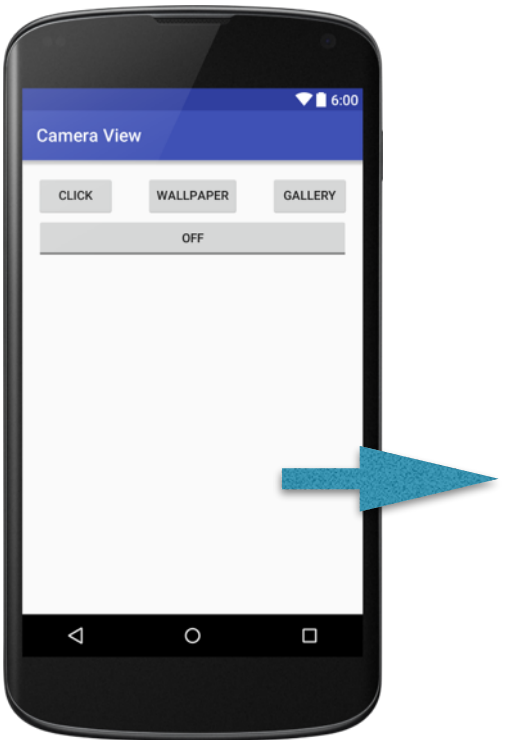
## 7.3 Running the Multithreading App

**Step 1:** Select File -> New -> Project -> Android Application Project (or) Android Project. Fill the forms and click "Finish" button.

**Step 2:** Open res -> layout -> activity_main.xml -> click Design -> Put the necessary components in the layout.

**Step 3:** Create a layout file for UI ( Design for UI after that the code will be generated automatically in activity_main.xml )

**activity_main.xml**




( image View Component also inserted )

**Step 4:** Right Click res -> New -> Android Resource directory -> select “raw” Resource type -> Ok

**Step 5:** Open res -> raw and add *.mp3 file

**Step 6:** Open java -> Main Activity.java and add following code

**MainActivity.java**

```
package com.example.kamarajios33.cameraview;

import android.content.Intent;

import android.database.Cursor;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.media.MediaPlayer;

import android.net.Uri;

import android.provider.MediaStore;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.ImageView;

import android.widget.Toast;

import android.widget.ToggleButton;



public class MainActivity extends AppCompatActivity {

    private static final int CAMERA_REQUEST= 1888;

    String p;

    ImageView i;

    Button bu,wall,b3;

    Bitmap photo;

    ToggleButton t;
```

```
    MediaPlayer m;
    int flag = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        i=(ImageView)findViewById(R.id.img);
        bu=(Button)findViewById(R.id.b1);
        wall=(Button)findViewById(R.id.wall);
        b3=(Button)findViewById(R.id.b3);
        t=(ToggleButton)findViewById(R.id.tb);
        bu.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent in = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(in,CAMERA_REQUEST);
            }
        });
        m = MediaPlayer.create(MainActivity.this,R.raw.sam);
        wall.setOnClickListener(new View.OnClickListener() {
@Override
            public void onClick(View v) {
                try {
                    if (flag == 1) {
                        getApplicationContext().setWallpaper(BitmapFactory.decodeFile(p));
                            Toast.makeText(getApplicationContext(), "Wallpaper Changed from
Gallery", Toast.LENGTH_SHORT).show();
                    } else if (flag == 2) {
                        getApplicationContext().setWallpaper(photo);
                            Toast.makeText(getApplicationContext(), "Wallpaper Changed from
Camera", Toast.LENGTH_SHORT).show();
                    }
```

```
else {
                        Toast.makeText(getApplicationContext(), "Wallpaper Not Set",
Toast.LENGTH_SHORT).show();
                }
             }
           catch (Exception e)
           {
              e.printStackTrace();
           }
         }
      });
      b3.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {

                                                    Intent in1 = new
Intent(Intent.ACTION_PICK,MediaStore.Images.Media.EXTERNAL_CONTENT_U
RI);
              startActivityForResult(in1,2);
         }
      });
      t.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            boolean checked = ((ToggleButton) v).isChecked();
  if(checked)
            {
               m.start();
               t.setText("Media Player On");
            }
```

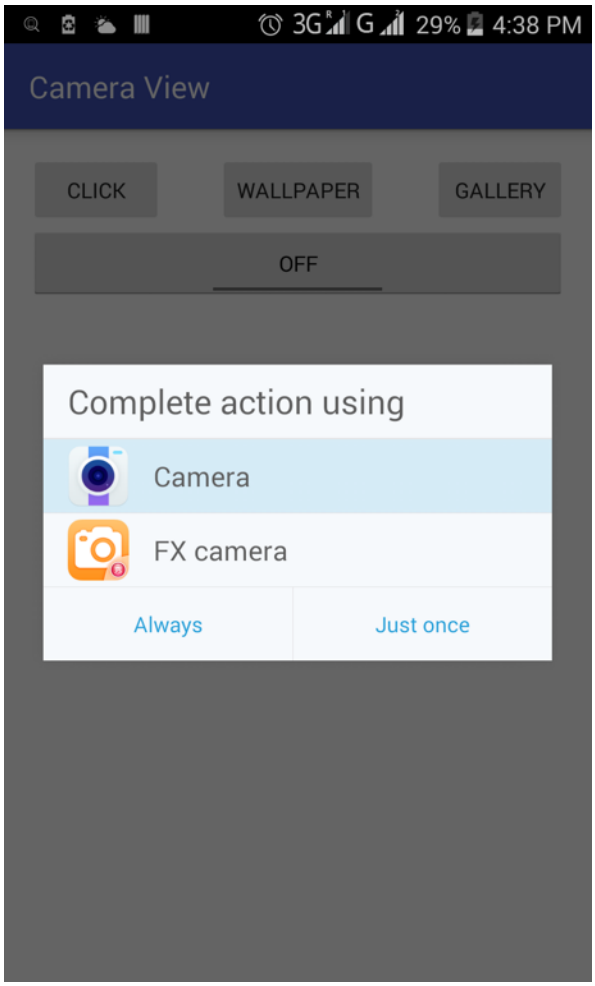
```
else              {
                  m.pause();
                  t.setText("Media Player Pause");
              }
          }
      });
   }
protected void  onActivityResult(int requestcode,int resultcode,Intent data)
{
   if(requestcode==CAMERA_REQUEST)
{
      photo    = (Bitmap)data.getExtras().get("data");
       i.setImageBitmap(photo);
       flag =2;
   }
   if(requestcode==2 && resultcode == RESULT_OK)
   {
      Uri sel = data.getData();
      String[] file ={MediaStore.Images.Media.DATA};
      Cursor c = getContentResolver().query(sel, file, null, null, null);
      c.moveToFirst();

      int co = c.getColumnIndex(file[0]);
      p = c.getString(co);
      c.close();
      i.setImageBitmap(BitmapFactory.decodeFile(p));
      flag = 1;
   }}}
```

**Step 7:** Open AndroidManifest.xml and add following code

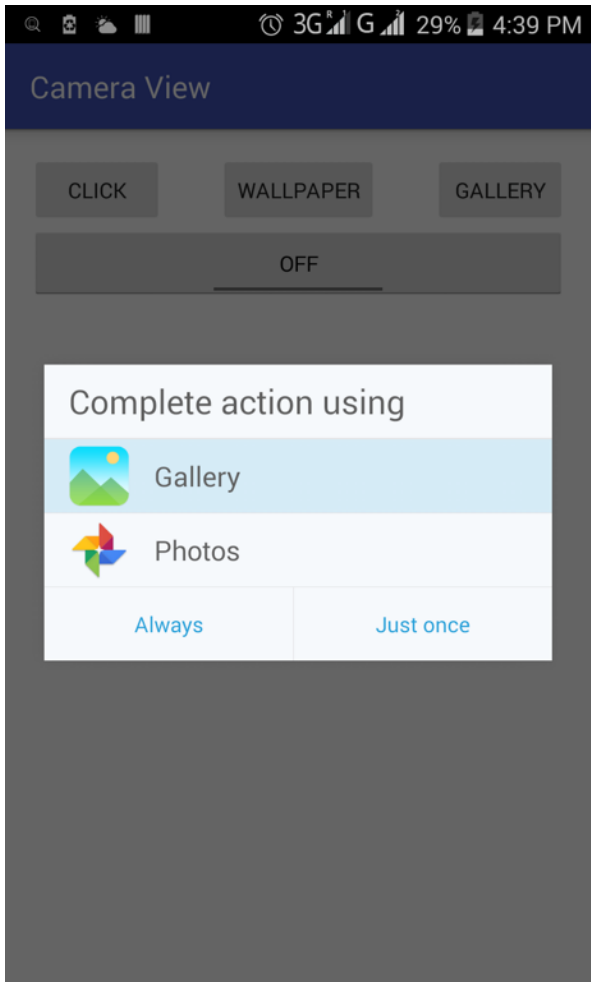
```
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

**Step 8:** The Output of the above code is as follows. As we have discussed, Multithreading App is easy.

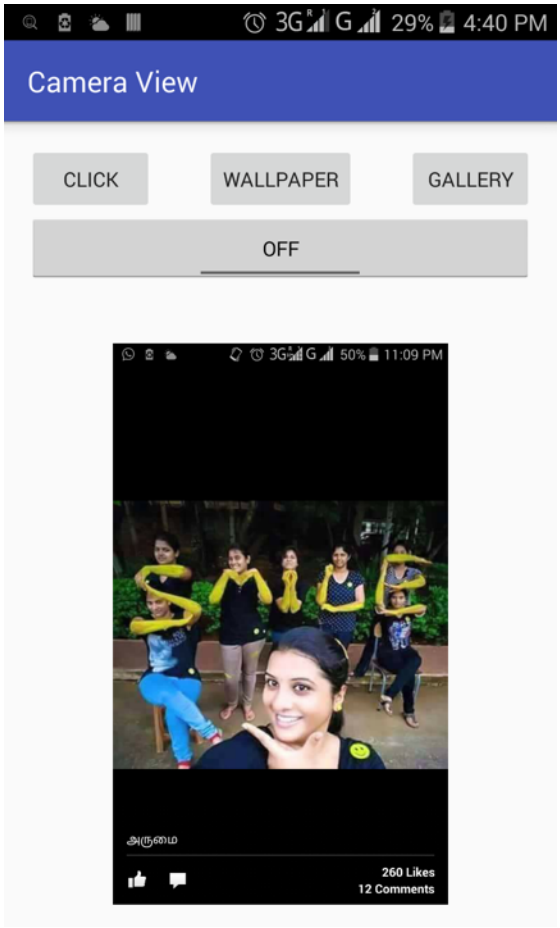
Click **CLICK** button



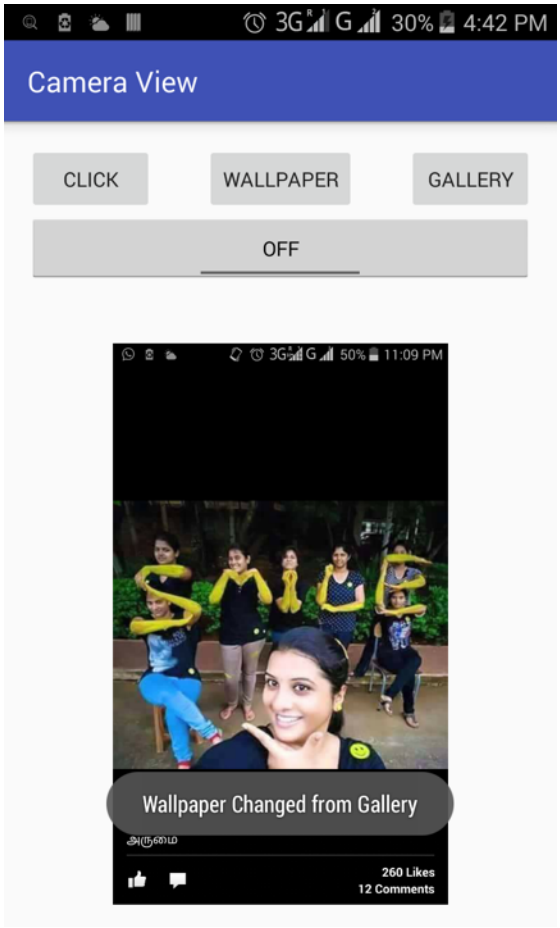

Click **GALLERY** button



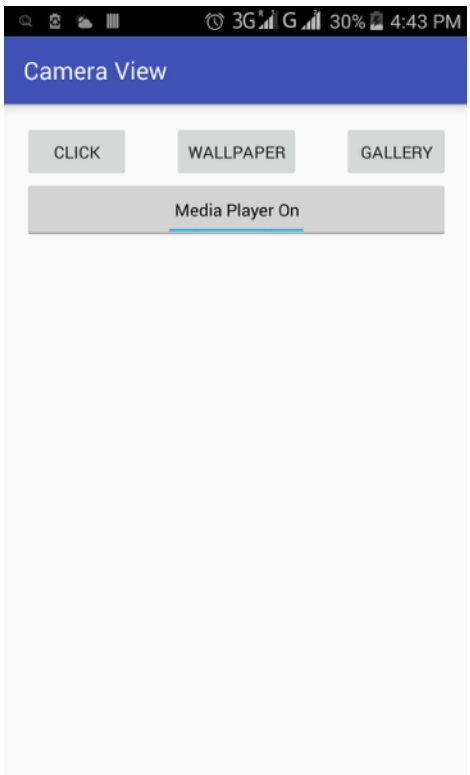
After click the **GALLERY** button

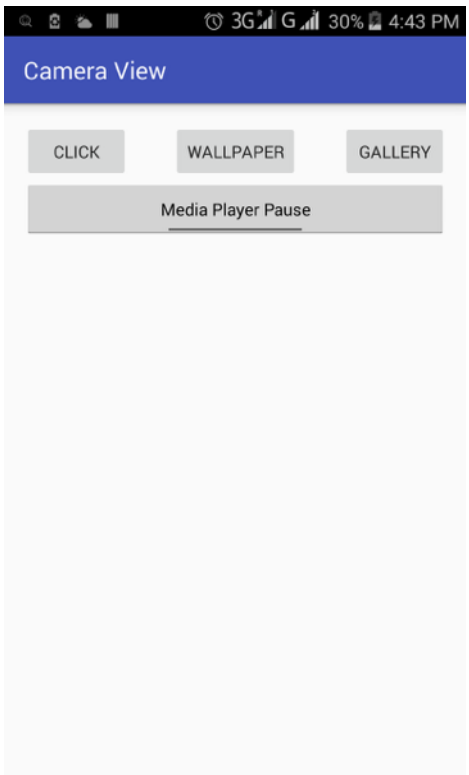



Click **WALLPAPER** button




Click Toggle (OFF) button



# 8. External data Storage

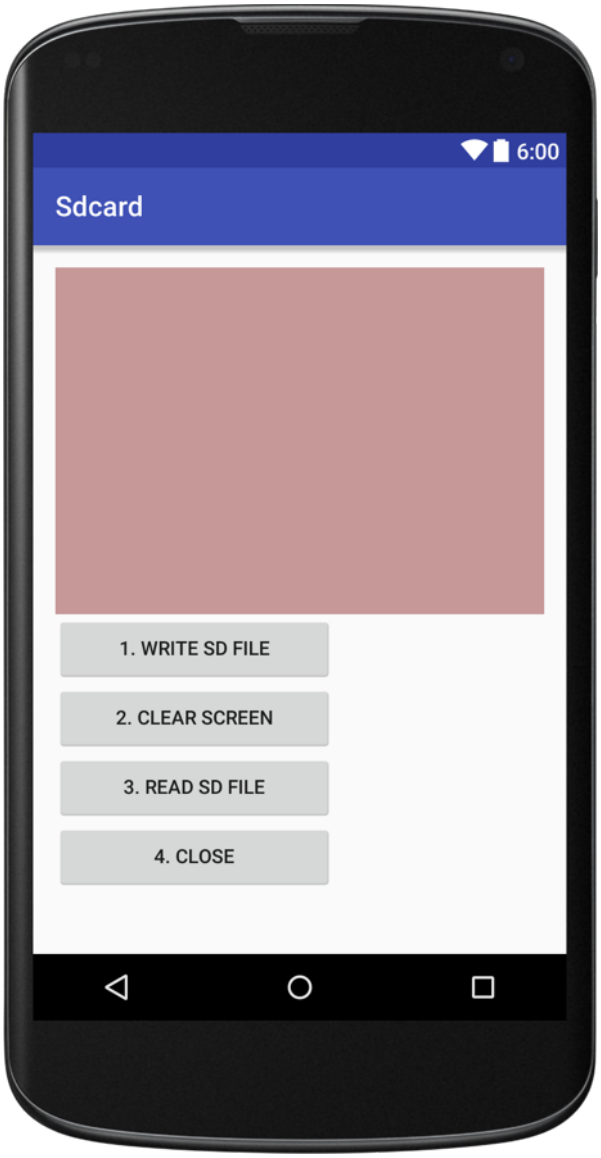
This Chapter to describe implement an application that writes data to the SD card

**Step 1:** Select File -> New -> Project -> Android Application Project (or) Android Project. Fill the forms and click “Finish” button.

**Step 2:** Open res -> layout -> activity_main.xml -> click Design -> Put the necessary components in the layout.

**Step 3:** Create a layout file for UI ( Design for UI after that the code will be generated automatically in activity_main.xml )

**Design - activity_main.xml**



**Step 4:** Create an activity file

**MainActivity.java**

```
package com.example.kamarajioslab01.sdcard;

import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

public class MainActivity extends AppCompatActivity {
    EditText txtData;
    Button btnWriteSDFile;
    Button btnReadSDFile;
    Button btnClearScreen;
    Button btnClose;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtData = (EditText) findViewById(R.id.txtData);
        txtData.setHint("Enter some lines of data here...");

        btnWriteSDFile = (Button) findViewById(R.id.btnWriteSDFile);
        btnWriteSDFile.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // write on SD card file data in the text box
```

```
      try {
         //File file= Environment.getExternalStorageDirectory();
         File myFile = new File(“/sdcard/mysdfile.txt");
         myFile.createNewFile();
         FileOutputStream fOut = new FileOutputStream(myFile);
         OutputStreamWriter myOutWriter =
               new OutputStreamWriter(fOut);
         myOutWriter.append(txtData.getText());
         myOutWriter.close();
         fOut.close();
         Toast.makeText(getBaseContext(),
               "Done writing SD 'mysdfile.txt'",
               Toast.LENGTH_SHORT).show();
      } catch (Exception e) {
         Toast.makeText(getBaseContext(), e.getMessage(),
               Toast.LENGTH_SHORT).show();
      }
   }// onClick
});// btnWriteSDFile

btnReadSDFile = (Button) findViewById(R.id.btnReadSDFile);
btnReadSDFile.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      // write on SD card file data in the text box
      try {
         File myFile = new File("/sdcard/mysdfile.txt");
         FileInputStream fIn = new FileInputStream(myFile);
         BufferedReader myReader = new BufferedReader(
               new InputStreamReader(fIn));
         String aDataRow = "";
         String aBuffer = "";
         while ((aDataRow = myReader.readLine()) != null) {
            aBuffer += aDataRow + "\n";
         }
         txtData.setText(aBuffer);
         myReader.close();
         Toast.makeText(getBaseContext(),
               "Done reading SD 'mysdfile.txt'",
               Toast.LENGTH_SHORT).show();
      } catch (Exception e) {
```

```
                Toast.makeText(getBaseContext(), e.getMessage(),
                Toast.LENGTH_SHORT).show();
            }
        }// onClick
    }); // btnReadSDFile

    btnClearScreen = (Button) findViewById(R.id.btnClearScreen);
    btnClearScreen.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            txtData.setText("");
        }
    }); // btnClearScreen

    btnClose = (Button) findViewById(R.id.btnClose);
    btnClose.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // clear text box
            finish();
        }
    }); // btnClose

}// onCreate

}// AndSDcard
```

**Step 5:** Open **AndroidManifest.xml** and add following code

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

**Step 6:** Run the Project. While running, the following output will be shown in the emulator.


3G G 61% 6:19 PM
Sdcard
john
1. WRITE SD FILE
2. CLEAR SCREEN
3. READ SD FILE
4. CLOSE
Done writing SD 'mysdfile.txt'


3G G 61% 6:20 PM
Sdcard
john
1. WRITE SD FILE
2. CLEAR SCREEN
3. READ SD FILE
4. CLOSE
Done reading SD 'mysdfile.txt'

# 9. Alert Upon receiving a message

## Objectives

- Dialogs, Notifications
- Running the Alert App

## 9.1 Dialogs, Notifications and Toasts

**Dialogs**

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

The `Dialog` class is the base class for dialogs, but we should avoid instantiating `Dialog` directly. Instead, ne of the following subclasses can be used.

1. `AlertDialog` - A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
2. `DatePickerDialog` or `TimePickerDialog` - A dialog with a pre-defined UI that allows the user to select a date or time.

These classes define the style and structure for a dialog, but the `DialogFragment` must be used as a container for our dialog. The`DialogFragment` class provides all the controls required to create a dialog, manage its appearance. Instead of calling methods on the `Dialog` object, the DialogFragment class and its methods can be used. Using `DialogFragment` to manage the dialog ensures that it correctly handles lifecycle events such as when the user presses the back button or rotates the screen. The `DialogFragment` class also allows to reuse the dialog's UI as an embeddable component in a larger UI, just like a traditional `Fragment`.

The AlertDialog class allows us to build a variety of dialog designs and is often the only dialog class we will need. There are three regions of an alert dialog which is shown below in the image:

1. Title
   This is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If we need to state a simple message or question (such as the dialog in figure 1), title is not required.
2. Content area
   This can display a message, a list, or other custom layout.
3. Action buttons
   There should be no more than three action buttons in a dialog.

The AlertDialog.Builder class provides APIs that allow you to create an AlertDialog with these kinds of content, including a custom layout. To build an AlertDialog, we can follow the steps given below.

1. Instantiate an AlertDialog.Builder with its constructor

```
AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
```

2. Chain together various setter methods to set the dialog characteristics

```
builder.setMessage(R.string.dialog_message).setTitle(R.string.dialog
_title);
```

3. Get the AlertDialog from create()

```
AlertDialog dialog = builder.create();
```

To add action buttons like those in figure 2, call the setPositiveButton() and setNegativeButton() methods:

```
AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
// Add the buttons builder.setPositiveButton(R.string.ok,
new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                        // User clicked OK button
                }
        });
builder.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                        // User cancelled the dialog
                }
        });
// Set other dialog properties
...
```

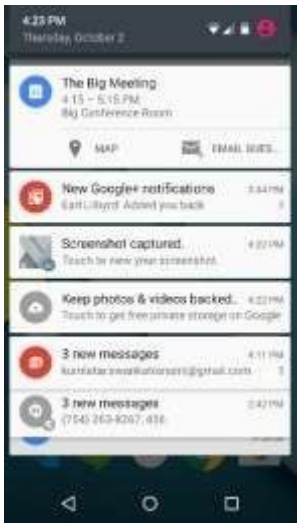
The set...Button() methods require a title for the button (supplied by a string resource) and a DialogInterface.OnClickListener that defines the action to take when the user presses the button. There are three different action buttons you can add:

- Positive - should be used to accept and continue with the action (the "OK" action).
- Negative - should be used to cancel the action.
- Neutral - should be used when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."

We can add only one of each button type to an AlertDialog. That is, we cannot have more than one "positive" button.

**Notifications**

A notification is a message which is displayed to the user outside of an application's normal UI. When we tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.





**Toasts**

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive.

For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.

```
Context context = getApplicationContext(); String text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration); toast.show();
```

The toast control receives the message to be displayed by makeToast() method. It will be visible only when the show() method is invoked. The above code can be combined into a single line of code as shown below.

```
Toast.makeText(this, "Heelo Toast ", Toast.LENGTH_SHORT).show();
```

## 9.2 Running the Alert App

**Step 1:** Select File -> New -> Project -> Android Application Project (or) Android Project. Fill the forms and click “Finish” button.

**Step 2:** Create an activity file

**MainActivity.java**

```
package com.example.user.alertdemo;
import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.view.Menu;
public class MainActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      AlertDialog.Builder builder = new AlertDialog.Builder(this);
        //Uncomment the below code to Set the message and title from the strings.xml
file
```

```
    //builder.setMessage(R.string.dialog_message) .setTitle(R.string.dialog_title);
    //Setting message manually and performing action on button click
    builder.setMessage("Do you want to close this application ?")
            .setCancelable(false)
            .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    finish();
                }
            })
            .setNegativeButton("No", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    //  Action for 'NO' Button
                    dialog.cancel();
                }
            });
    //Creating dialog box
    AlertDialog alert = builder.create();
    //Setting the title manually
    alert.setTitle("Alertdemo");
    alert.show();
    setContentView(R.layout.activity_main);
}}
```

**Step 3:** Run the Project. While running, the following output will be shown in the emulator.


# 10. Alarm Clock

## Objectives

After completing this chapter we will be able to understand the following topics.

- Introduction to Services
- Lifecycle of a Service
- Managing Services
- Introduction to Broadcast Receivers
- Types of Broadcast Receivers
- Creating and Registering Broadcast Receivers

## 10.1 Introduction to Services

Service can be described as long running background app without any user interface (UI). The services will be running as a separate thread, while any other activity and task is running on fore ground. It can be managed by app component such as activity.

While developing android app, we should be able to manage different task simultaneously. For example, playing music in background while playing games. Such simultaneous tasks are managed by services. To interact with services we need to create an activity. For example, service might interact with a content provider, music player and so on. All these tasks will be running in the background.

### 1. Forms of Services

Services are used for regularly updating the data source in background. The updated data source is used by activities running on foreground. Services are mainly used for triggering notifications such as email or message arrival. Services are controlled (i.e. Started/stopped) from other android app components such as broadcast receivers, activities and other service components. Services are classified into two types. They are

- ➢ Unbound Services
- ➢ Bound Services

a) **UnBound Service**

The unbound service is also known as started service. The startService() method is used to start the service,onces the service is started it will be running in background for indefinite period of time, even if we destroy the started component. Started service performs the intended operation and does not return any result to the caller component.

Example:
When we open the game, music starts playing in background automatically till we exit the game.

b) **Bound Services**

Bound Service can be referred as client–server approach . The bound services allow the app components to interact with each other. It allows the appl components to send request and getresult using inter process communication(IPC). Bound service uses bindService() method to bind with one or more app components. Bound service will be running until another component is bound to it. Multiple components can be binded to the service at a time. The service will be destroyed when the bind components are unbind.

## 2. Lifecycle of a Service

The lifecycle of a service can follow two different paths depending on the form of the service, which can be either unbound or bound.

To understand the lifecycle of a service, consider the example like playing games. When we open the game app, an activity is started. During the lifetime of this app, activity may invoke a service.
i.e. service to enable or disable the music by changing the game or exiting the game. If user exits the game app, all service components acquired by the app are released and destroyed.

### Lifecycle Methods

Some of the important callback methods that you can override are:

| Method | Syntax and Description |
|---|---|
| **onStartCommand()** | ***public int onStartCommand(Intent intent, int flags, int startld)***<br>This method is called when another app component request to start service by calling the startService() method |
| **onBind()** | ***public abstract IBinder onBind(Intent intent)***<br>This method is called when another component requests to bind with the service by calling the bindService() method |
| | ***public void onCreate()*** |

| | |
|---|---|
| **onCreate()** | This method is called when the service is first created |
| **onUnbind()** | ***public boolean onUnbind(Intent intent)***<br>This method is called when all components bound to the service have been unbound by calling the unbindService() method |
| **onDestroy()** | ***public void onDestroy()***<br>This method is called when the service is no longer being used and is being destroyed |

### Lifecycle of UnBound (Started) Service

To invoke the service component startService() method need to be called. Once the service is started, it is running in background for indefinite period of time :

- The service stops itself by calling the stopSelf() method.
- Another app component stops the service by calling the stopService() method. When a service is stopped, when the system destroys service components and its resources.

The figure depicts the Callback method Executed during the Lifecycle of a started service or unbound service.

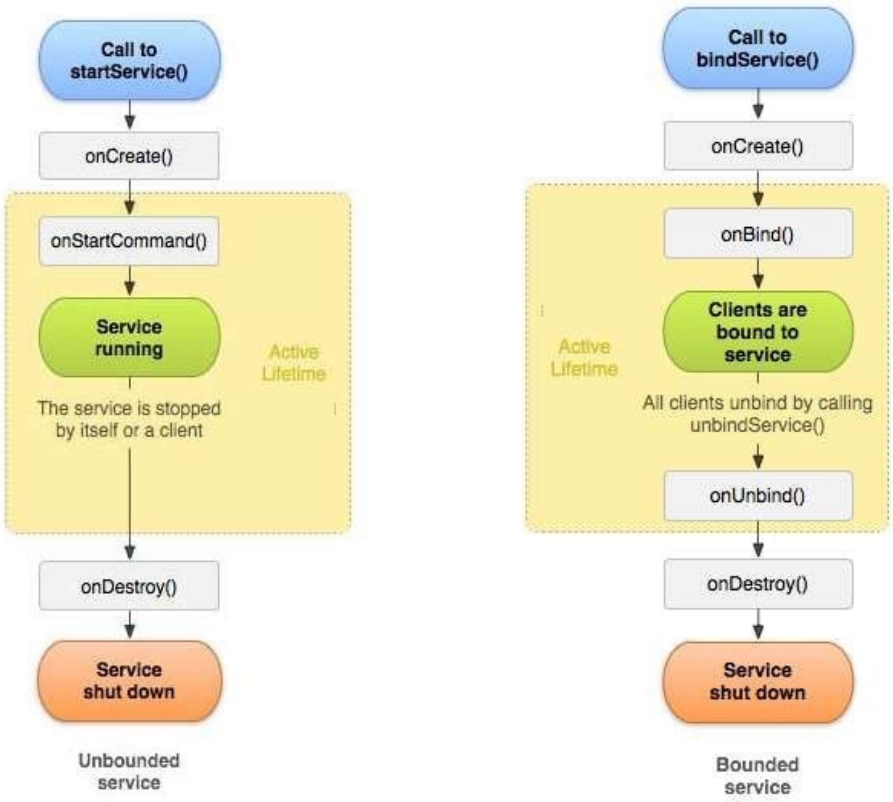
### Lifecycle of Bound Service

We can create bound service from another components by calling bindService() method. While calling the bindService() method, the service is bind to the components. We can bind multiple components with a service. Bound services communicate through IBinder interface.

It can be unbind by calling unbindService() method. Service need not to be explicitly stopped, it is automatically stopped and destroy all the resources which are bind to that service component. Stopping services automatically, is a good practice in app development because if a service is not stopped itself, optimization of resources cannot be achieved.

For example, when a started service is downloading a file over a network, the service should stop itself when the operation completed.

The following figure depicts the callback method executed during the lifecycle of a bound service.




## Managing Services

### 1. Creating a Service

Service can be created in two ways,

1. Explicitly creating the service.
2. Through an app component that requests to connect to the service.

Services are independent blocks of code to perform given tasks. The services are getting the latest updated date from the other services which are running in the background. To create a service we need to extend the Service class and override the some of the callback methods available in service class. The class Service is available in the package 'android.app '. The callback methods of services handle the lifecycle of the service that has been created. It also allows the other service components to connect to this newly created service by calling the bindService().

### 2. Starting a Service

To implement services in the Android apps, we need to:

- Create a service by extending the android. app. Service class.
- Register the service in the AndroidManifest.xml file.

Regardless of the service types whether it is bound or unbound serive, alwaya the startService() callback method is invoked to start the service. This method takes Intent object as parameter, which contains the info about the service need to be started.

The following code shows how to start a service.

```
Intent i = new intent(this, myService.class);
startService(i);
```

When the service component call the startService() method, it will check whether the service is running or not, if service is not running the android system will call onCreate() method, if service is already running in background then android system will call onStartCommand() method.When multiple components start a service, thay call onStartcommand() method then each start request is called individually.

The service will start running after receiving request from the component. It will continue running, until it stopSelf() or stopService() method is called. We have to note one point that even multiple requests made to the startService() method, the call to the startService() method will not be nested. No matter how many requests or calls to the startService() method made, but the service will be stopped once the stopService() or stopSelf() method is called. . In bound service, the component that starts the service can create a PendingIntent[it is a token that is given to another app (e.g. NotificationManager, AlarmManager, Home Screen AppWidgetManager), which allows that app to use current app's permissions to execute a predefined piece of code] object and pass this intent object to the caller service components. The service can then use this object to send a result back to the calling component.

## 3. Stopping a Service

As discussed previously service is independent block of code, it needs to be manage its own lifetime. The service will be running in the background until android platform regain the memory occupied by the service. When service is ideal for long time, it has to be stopped by itself by calling stopSelf() callback method. To destroy the service explicitly we need to call the stopServcie(Intent Service) callback method. Unlike the onStartCommand() method, only one call to the stopService or stopSelf() method destroys the service.

**Example**

This example will take us through simple steps to show how to create our own Android Service.

| No | Steps |
|---|---|
| 1 | Create an android project in android studio. |
| 2 | Define the service in *AndroidManifest.xml* file using <service.../> tag. |

| | |
|---|---|
| 3 | Create a Service Class. |
| 4 | Create layout file. Define the required strings in *res/values/strings.xml* file |
| 5 | Create an Activity class. |
| 6 | Run the app to launch Android emulator and verify the result of the changes done in the app. |

## 10.1 Introduction to Broadcast Receivers

In an android device, broadcast message is generated, to inform the currently running app that certain event has occurred.

For example, an app initiates broadcast message to another app stating that some data has been downloaded to the device and available for use. Broadcast receiver will intercept this communication and will initiate appropriate action.

Broadcasts can be generated in two ways:

- **The system-level event broadcast:**
  The broadcast message generated by the system is known as system-level event broadcast, such as the screen is being turned off, the battery is low, or an SMS has arrived.
- **An app-level event broadcast:**
  The broadcast message generated by an app is known as app-level event broadcast, For example, an app that is downloading some data may want to inform other apps that the data has been downloaded and ready to use. By using app-level event broadcast, we can inform other apps that some event has occurred. This can be achieved using a broadcast receiver, which enables the apps to register themselves to receive a notification whenever a new feed is available.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents:

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

## Types of Broadcast Receivers

Broadcasts are majorly classified as:

- Normal Broadcasts
- Ordered Broadcasts

### 1. Normal Broadcasts

The broadcasts messenger sends message in undefined order to all the registered receivers at the same time. For example, the broadcasts, such as battery low (ACTION_BATTERYLOW) or timezone changed (ACTION_TIMEZONE_CHANGED) will be received by all the registered receivers at the same time. This type of broadcasts are sent using the sendBroadcast() method of the android.content.Context class.

**Syntax**

```
public abstract void sendBroadcast(Intent intent, String receiverPermission)
```

- **The fisrt parameter is Intent** object that has to be broadcasted. All broadcast receivers with intent filters matching this intent will receive thes broadcast.
- **The second parameter is receiverPermission. It** is an optional parameter that specifies a permission that a broadcast receiver must have in order to receive the broadcast.

### 2. Ordered Broadcasts

The broadcast messenger sends message to all the registered receivers in an ordered manner, which means that a broadcast is delivered to one receiver at a time. When a receiver receives a broadcast, it can either propagate the broadcast to the next receiver or it can completely abort the broadcast. If a broadcast message is aborted by a receiver, it will not be passed to other receivers.

This type of broadcasts are sent using the sendOrderedBroadcast() method of the android.content.BroadcastReceiver class.

**Syntax**

```
public abstract void sendOrderedBroadcast (Intent intent, String receiverPermission)
```

The parameters are similar to the sendBroadcast() method.

## Creating and Registering a Broadcast Receiver

### 1. Creating the Broadcast Receiver

Each broadcast is delivered in the form of an intent object. Broadcast receivers enable apps to receive intents object that are either broadcasted by the system or by other apps. A broadcast receiver is implemented as a subclass of Broadcast Receiver class and overriding the onReceive() method where each message is received as a Intent object parameter.

The onReceive() method is called when a broadcast receiver receives a broadcast intent object. This method must include the code that needs to be executed when a broadcast is received. This method is usually called within the main thread of its process. Therefore, we should not perform long-running operations in this method. The system allows a timeout of 10 seconds before considering the receiver to be blocked. The system can kill a receiver that has been blocked. Also, we cannot launch a pop-up dialog in our onReceive ( ) method implementation. The signature of the onReceive () method is:

```
public abstract void onReceive (Context context, Intent intent)
```

where,

- **context** is the context in which the receiver is running.
- **intent** is the Intent object being received.

## 2. Registering a Broadcast Receiver

To register the broadcast receiver, app's broadcast intent has to be registered in AndroidManifest.xml file.

To declare a broadcast receiver in the manifest file, you need to add a <receiver> element as a child of the <application> element by specifying the class name of the broadcast receiver to register. The <receiver> element also needs to include an <intent-filter> element that specifies the action string being listened for.

Broadcaster receivers can be registered by either one the two methods.

- Statically
- Dynamically

By using the method registerReceiver() a broadcast receiver is registered dynamically. If the manifest file is used for registering the broadcast receiver, it is known as static method.

### Static Registering

**Example:**

```
<application android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <receiver android:name="MyReceiver">
        <intent-filter>
            <action
android:name="android.intent.action.BOOT_COMPLETED">
        </action>
        </intent-filter>
    </receiver>

</application>
```

There are several system generated events defined as final static fields in the Intent class. The following table lists a few important system events.

### Dynamic Registering

We can register a broadcast receiver using the registerReceiver() method. The registerReceiver() method is called with a broadcast intent that matches the filter in the main app thread. The signature of the registerReceiver() method is :

```
public abstract Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)
```

where,

- **receiver** describes the broadcast receiver to handle the broadcast.
- **filter** describes the intents that the broadcast receiver can receive.

The following code snippet shows how to register a broadcast receiver dynamically:

```
IntentFilter filter = new IntentFilter()
filter.addAction(ConnectivityManager.CONNECTIVITY_ACTION);
registerReceiver(myBroadcastReceiverl,filter);
```

## 3. Listening to Broadcast Events

By broadcasting an event using intents, we can make our app to react to events without modifying the actual code. The app can replace or enhance the core app functionality or react to the system changes and app events by listening to a broadcast message. For example, by listening for the incoming call broadcast, an app can modify the ringtone or volume based on the caller.

To broadcast an event using intents, we need to create an Intent object and specify its action, data, and category components in such a way that the registered broadcast receivers are able to identify the event accurately.

The following code snippet illustrates the broadcasting of an event using intents:

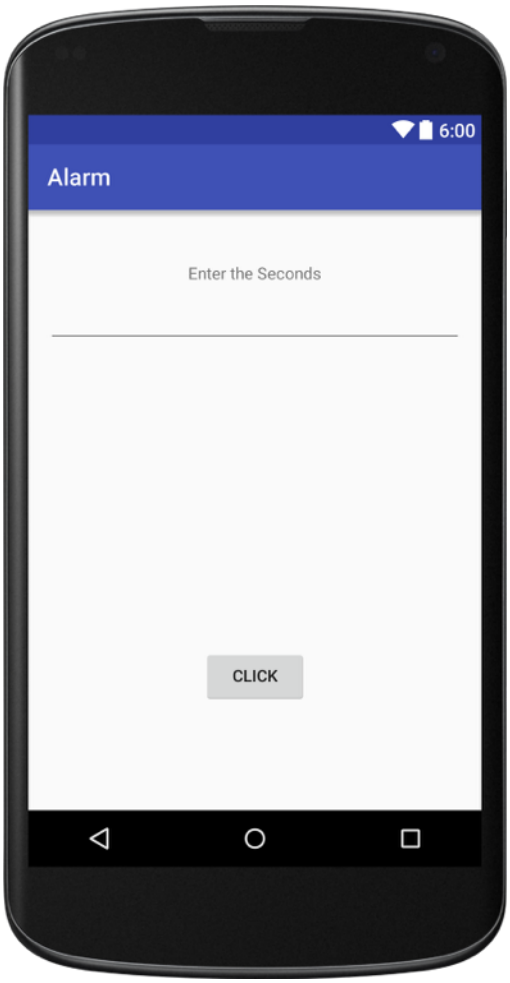
```
Public static final String MY_EVENT =
“com.broadcast.demo.action.NEWBAOADCASTEVT”;
Intent i = new Intent(MYEVENT);
// Additional information for the intent. i.putExtra("eventName", eventType);
sendBroadcast(i);
```

## 10.2 Running the alarm Clock App

**Step 1:** Create an android project using android studio

**Step 2:** Create a layout file for UI.

**activity_main.xml** ( After design, the code will be generated automatically in activity_main.xml file)




**Step 3:** Right Click res -> New -> Android Resource directory -> select “raw” Resource type -> Ok

**Step 4:** Open res -> raw and add *.mp3 file

**Step 5:** After creating the layout, open the manifest file named AndroidManifest.xml. Define the receiver using the <receiver> tag.

The code for **AndroidManifest.xml** is given below.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.kamarajios33.alaram" >

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      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>
      <receiver android:name="alaram"/>
   </application>

</manifest>
```

**Step 6:** Create a alaram class by receiving the broadcast service.

**alaram.java**

```
package com.example.kamarajios33.alaram;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.MediaPlayer;
import android.widget.*;
/**
 * Created by kamarajios33 on 13/11/15.
 */
public class alaram extends BroadcastReceiver {
   MediaPlayer m;
   public void onReceive(Context context,Intent intent)
```

```
{
      m = MediaPlayer.create(context,R.raw.cine1);
      m.start();
   Toast.makeText(context, "Alarm....Get up", Toast.LENGTH_LONG).show();
}}
```

**Step 7:** Create an activity file.

**MainActivity.java**

```
package com.example.kamarajios33.alaram;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   Button b1;
   TextView t1;
   EditText e1;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      b1=(Button)findViewById(R.id.b1);
      t1=(TextView)findViewById(R.id.t1);
      e1=(EditText)findViewById(R.id.e1);
      b1.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            startalert();}
      });}
   public void startalert()
   {
      int i = Integer.parseInt(e1.getText().toString());
```

```
        Intent in = new Intent(this,alaram.class);
        PendingIntent pendingIntent = PendingIntent.getBroadcast(getApplicationContext(),
900000,in,0);
        AlarmManager
alarmManager=(AlarmManager)getSystemService(ALARM_SERVICE);
        alarmManager.set(AlarmManager.RTC_WAKEUP,System.currentTimeMillis()+
(i*1000),pendingIntent);
        Toast.makeText(getApplicationContext(),"Alaram set in " + i + "
Seconds",Toast.LENGTH_LONG).show();

    }
}
```

**Step 8:** Run the Project, While running, the following output will be shown in emulator.


# 11. GPS Location Information

## Objectives

After completing this chapter we will be able to understand the following topics.

- What is GPS
- Android Location API

## 11.1 What is GPS

One of the unique features of mobile applications is location awareness. Mobile users take their devices with them everywhere, and adding location awareness to your app offers users a more contextual experience. The location APIs available in Google Play services facilitate adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

Android devices use the same global positioning technology as Google Maps and most third-party GPS tools do. This allows users to locate themselves on a map, find and navigate to destinations via detailed directions, and search maps using a number of different methods. There are two ways to get Android GPS service. They are

- Android Location API
- Google Maps API

**Android Location API**

The location APIs make easier the job of building location-aware applications, without focusing on the details of the underlying geo location positioning technology. Android contains the android.location package which provides the API to determine the current geo position.

**Google Maps API**

With the Google Maps Android API, we can add maps to our app that are based on Google Maps data. The API automatically handles access to Google Maps servers, data downloading, map

display, and touch gestures on the map. We can also use API calls to add markers, polygons and overlays, and to change the user's view of a particular map area.

The key class in the Google Maps Android API is MapView. A MapView displays a map with data obtained from the Google Maps service. When the MapView has focus, it will capture keypresses and touch gestures to pan and zoom the map automatically, including handling network requests for additional maps tiles. It also provides all of the UI elements necessary for users to control the map. Our application can also use MapView class methods to control the map programmatically and draw a number of overlays on top of the map.

## 2. Android Location API

The android.location API is provided by android for implementing location based services in android app. The following classes are available in the andoid.location API and will be helpful while using the location based services.

**Location**
The Location class represents a geographic location. A location can consist of a latitude, longitude, timestamp, and other information such as bearing, altitude and velocity.

**LocationListener**
Used for receiving notifications from the LocationManager when the location has changed.

**LocationManager**
The LocationManager class provides access to the Android location service. This services allows to access location providers, to register location update listeners and proximity alerts and more.

**LocationProvider**
The LocationProvider class is the superclass of the different location providers which deliver the information about the current location. This information is stored in the Location class.

**Address**
Address is a class representing an address, i.e, a set of Strings describing a location. The addres format is a simplified version of xAL (eXtensible Address Language)

**GeoCoder**
Using the Geocoder class in the Android framework location APIs, you can convert an address to the corresponding geographic coordinates. This process is called geocoding. Alternatively, you can convert a geographic location to an address. The address lookup feature is also known as reverse geocoding.

The Android device might have several LocationProvider available and you can select which one you want to use. In most cases you have the following LocationProvider available.

**Using GPS and setting the current location**

The user can check whether the GPS is enabled or not. We can find out, if a LocationManager is enabled via the isProviderEnabled() method. If it iss not enabled we can send information to the user. This can be achieved via an Intent with the action called Settings. ACTION_LOCATION_SOURCE_SETTINGS for the android.provider.Settings class. Which represents settings in an android device.

```
LocationManager service = (LocationManager)
getSystemService(LOCATION_SERVICE);
boolean enabled = service
    .isProviderEnabled(LocationManager.GPS_PROVIDER);

// check if enabled and if not send user to the GSP settings
// Better solution would be to display a dialog and suggesting to
// go to the settings if (!
enabled) {
    Intent intent = new
Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
```

Typically you would open an AlarmDialog prompt the user and if he wants to enable GPS or if the application should be canceled. You cannot enable the GPS directly in your code, the user has to do this.

**Step 1:**

To use GPS in your application first of all you must specify the uses-permission in Android manifest file:

AndroidManifest.xml

```
<manifest>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"></uses-
permission>
```

**Step 2: Create a Activity Layout**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/andro id"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:orientation="vertical" >
<TextView android:id="@+id/gps_text"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello" />
</LinearLayout>
```

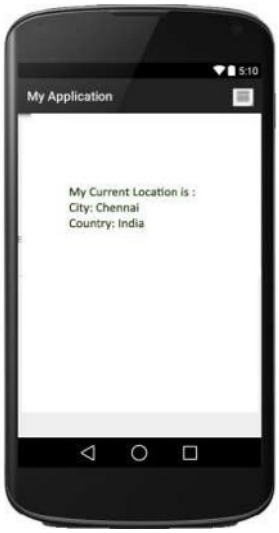
**Step 3: Create a Java Class file**

**CurrentLocationActivity.java**

```
import java.io.IOException;
 import java.util.List;
import java.util.Locale;

import android.app.Activity;
import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class CurrentLocationActivity extends Activity {
       /** Called when the activity is first created. */ Private TextView


  super.onCreate(savedInstanceState); setContentView(R.layout.main);

  gpsLocationView=(TextView) findViewById(R.id.gps_text);

        /* Use the LocationManager class to obtain GPS locations */

  LocationManager mlocManager =
  (LocationManager)getSystemService(Context.LOCATION_SERVICE);

  LocationListener mlocListener = new MyLocationListener();

  mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0,
  0, mlocListener);
        }

        /* Class My Location Listener */



  public class MyLocationListener implements LocationListener
       {

                @Override
  public void onLocationChanged(Location loc)
  {
  loc.getLatitude();
  loc.getLongitude();

  Geocoder gcd = new Geocoder(getApplicationContext(),
  Locale.getDefault());
  Try
```

```
 {
addresses = gcd.getFromLocation(loc.getLatitude(),loc.getLongitude(), 1);

}
catch (IOException e)
 {
                        // TODO Auto-generated catch block
e.printStackTrace();

 }

String text=(addresses!=null)?"City : "+addresses.get(0).getSubLocality()+"\n Country
: "+addresses.get(0).getCountryName():"Unknown Location";

   String locationValue = "My current location is: "+ text;
gpsLocationView.setText(locationValue);
            }

     @Override
public void onProviderDisabled(String provider)
{
Toast.makeText(getApplicationContext(),"Gps
Disabled",Toast.LENGTH_SHORT ).show();

            }

            @Override
public void onProviderEnabled(String provider)
{
Toast.makeText(getApplicationContext(),"Gps
Enabled",Toast.LENGTH_SHORT).show();
            }

            @Override
public void onStatusChanged(String provider, int status, Bundle extras)
{

 }

 }
}
```

## Output



# 13. Testing and Publishing

## Objectives

After completing this chapter we will be able to understand the following topics.

- Debugging
- Android App Development Life Cycle
- APK Conversion Process
- App Publishing Guidance
- Tips for Launching an App

## 1. Debugging

The Android SDK provides most of the tools that you need to debug your applications. You need a JDWP-compliant debugger if you want to be able to do things such as step through code, view variable values, and pause execution of an application. If you are using Android Studio, a JDWP- compliant debugger is already included and there is no setup required. If you are using another IDE, you can use the debugger that comes with it and attach the debugger to a special port so it can communicate with the application VMs on your devices. The main components that comprise a typical Android debugging environment are:

adb

adb acts as a middleman between a device and your development system. It provides various device management capabilities, including moving and syncing files to the emulator, running a UNIX shell on the device or emulator, and providing a general means to communicate with connected emulators and devices.

Dalvik Debug Monitor Server

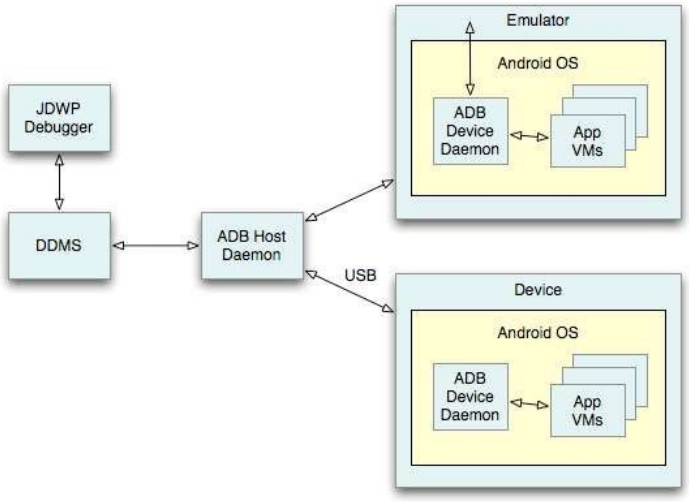
DDMS is a graphical program that communicates with your devices through adb. DDMS can capture screenshots, gather thread and stack information, spoof incoming calls and SMS messages, and has many other features.

Device or Android Virtual Device

Your application must run in a device or in an AVD so that it can be debugged. An adb device daemon runs on the device or emulator and provides a means for the adb host daemon to communicate with the device or emulator.

JDWP debugger

The Dalvik VM (Virtual Machine) supports the JDWP protocol to allow debuggers to attach to a VM. Each application runs in a VM and exposes a unique port that you can attach a debugger to via DDMS. If you want to debug multiple applications, attaching to each port might become tedious, so DDMS provides a port forwarding feature that can forward a specific VM's debugging port to port 8700. You can switch freely from application to application by highlighting it in the Devices tab of DDMS. DDMS forwards the appropriate port to port 8700. Most modern Java IDEs include a JDWP debugger, or you can use a command line debugger such as jdb.



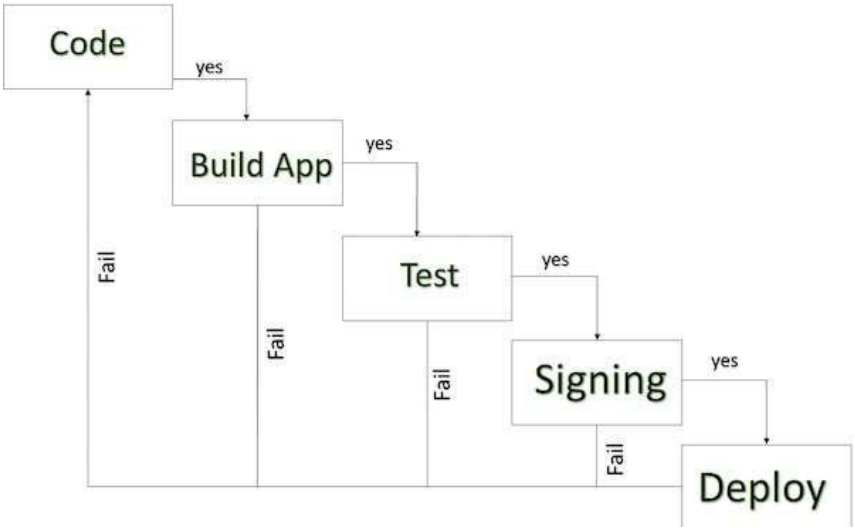

**DDMS**

Android ships with a debugging tool called the Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more. This page provides a modest discussion of DDMS features; it is not an exhaustive exploration of all the features and capabilities. The following operation done using DDMS

- Viewing heap usage for a process
- Tracking memory allocation of objects
- Working with an emulator or device's file system
- Examining thread information
- Starting method profiling
- Network Traffic tool
- LogCat
- Emulating phone operations and location

## 2. Android App Development Life Cycle

Android application publishing is a process that makes your android applications available to users. Infact, publishing is the last phase of the android application development process.



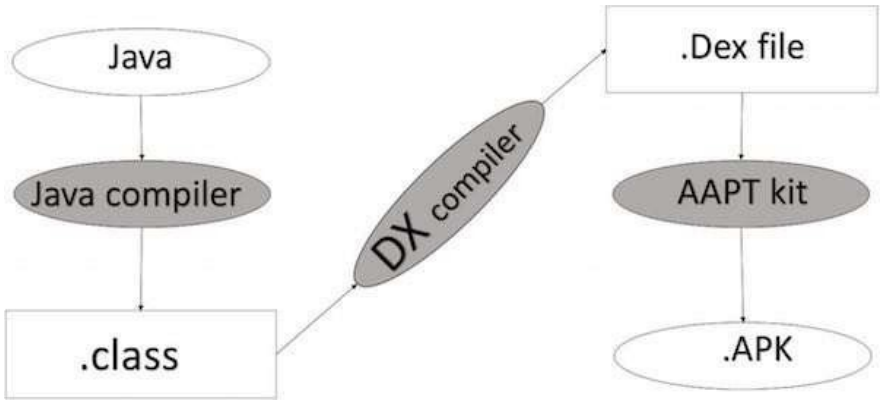

## 3. APK Conversion Process

Before exporting the apps, you must be aware of some of the tools like

- **Dxtools**(Dalvik executable tools ): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

**Export Android Application Process**

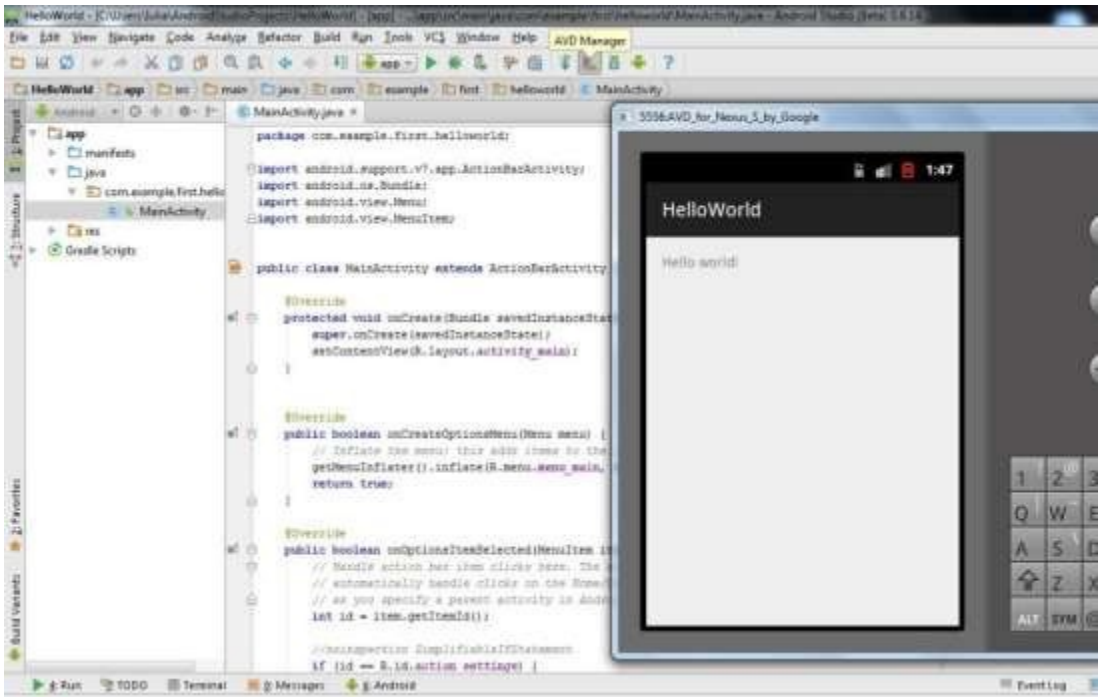



To export an application, just open that application project in Android studio and selectBuild → Generate Signed APK from your Android studio and follow the simple steps to export your application.

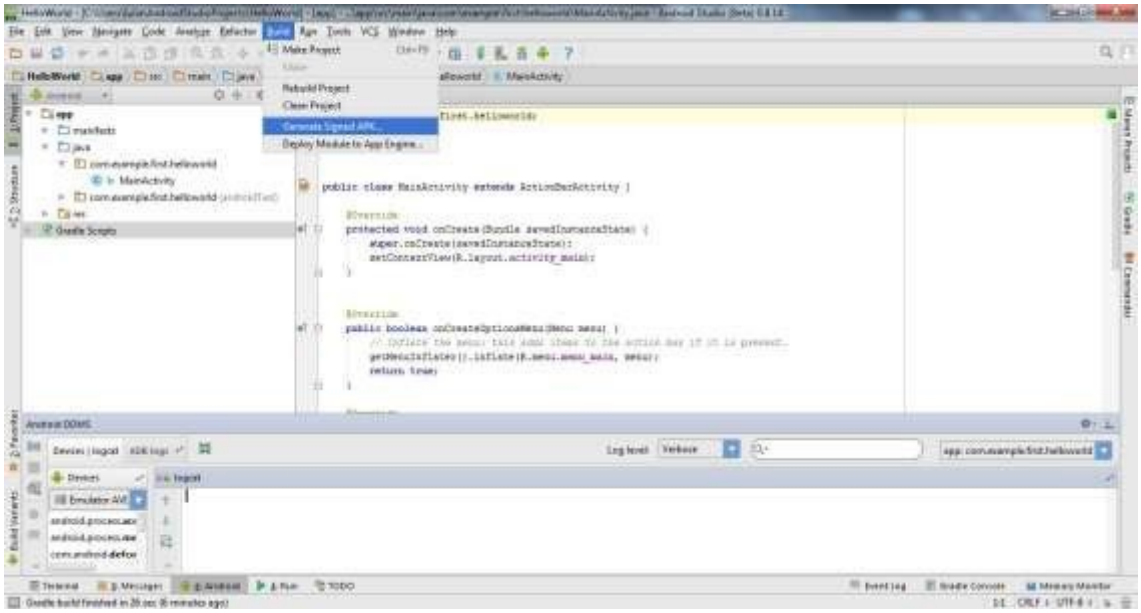
Exporting (generating signed APK) your application from Android Studio for installing on mobile devices. Application is exported with apk extension.

**Example**

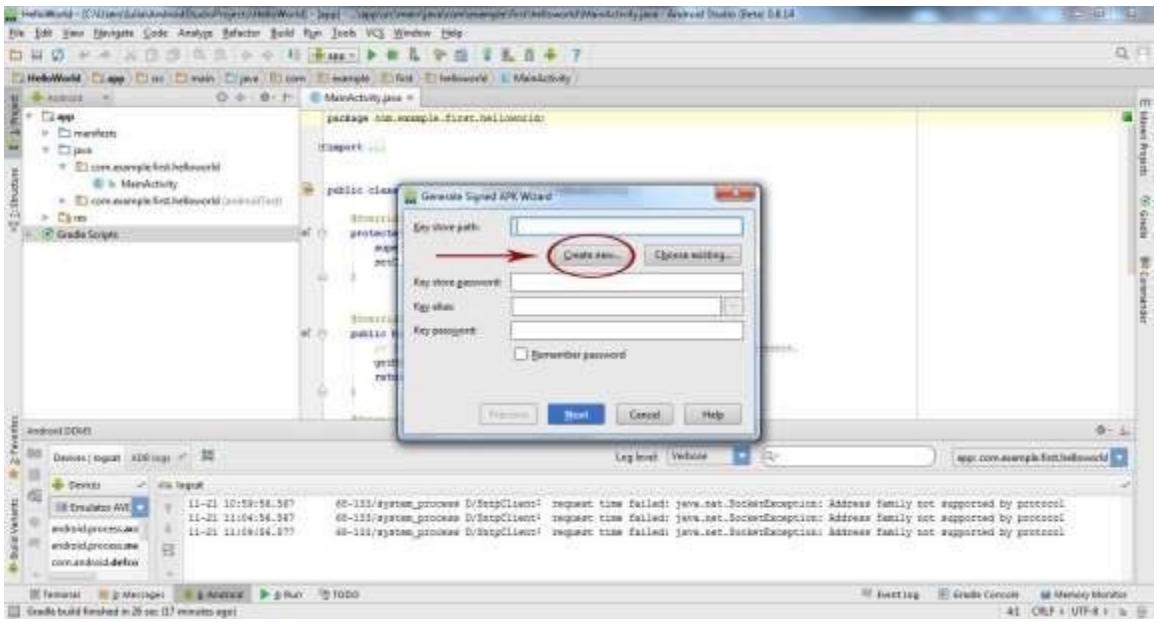
a. Open Android Studio and first project Hello World

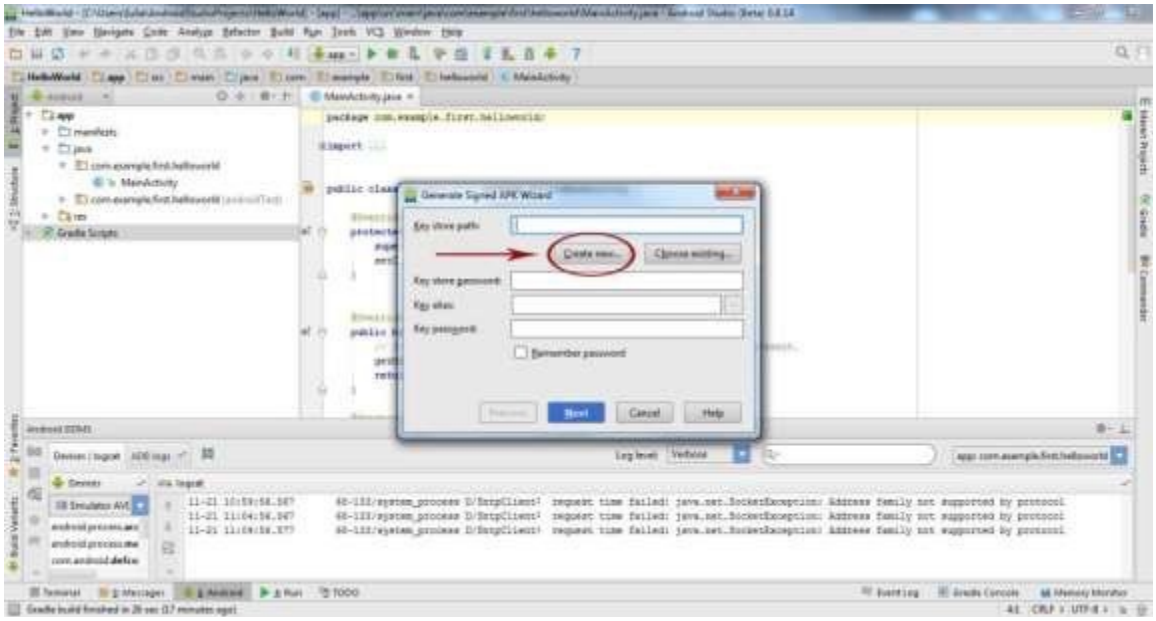



b. Choose from menu: **Build** -> **Generate Signed APK…**

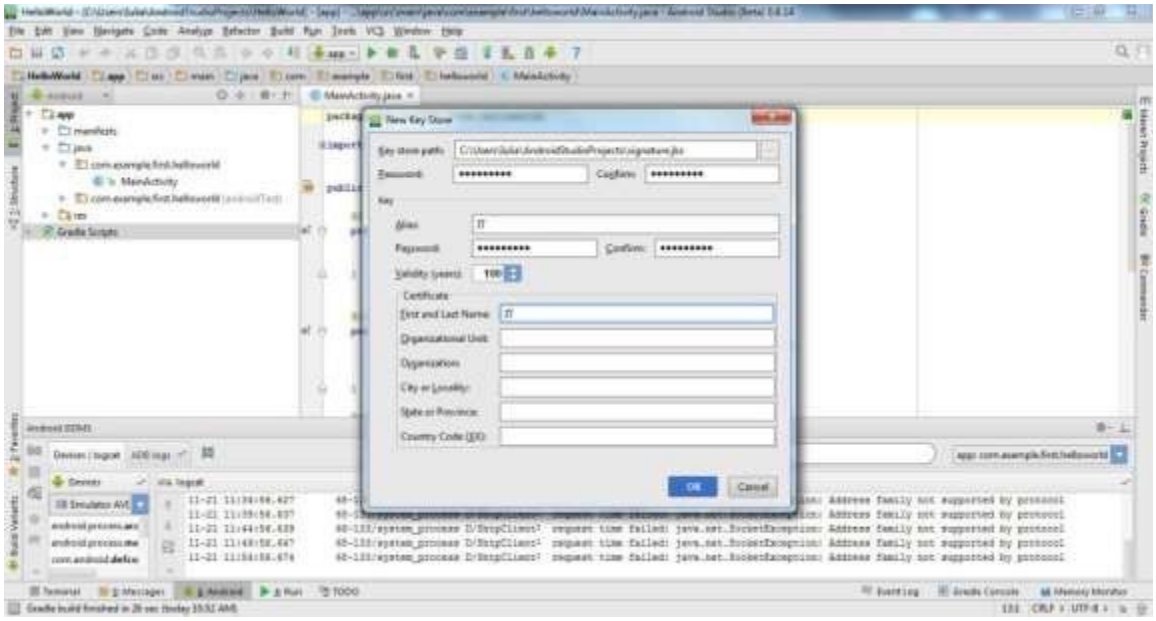


c. In next window Generate Signed APK Wizard on Key store path chooseCreate new…

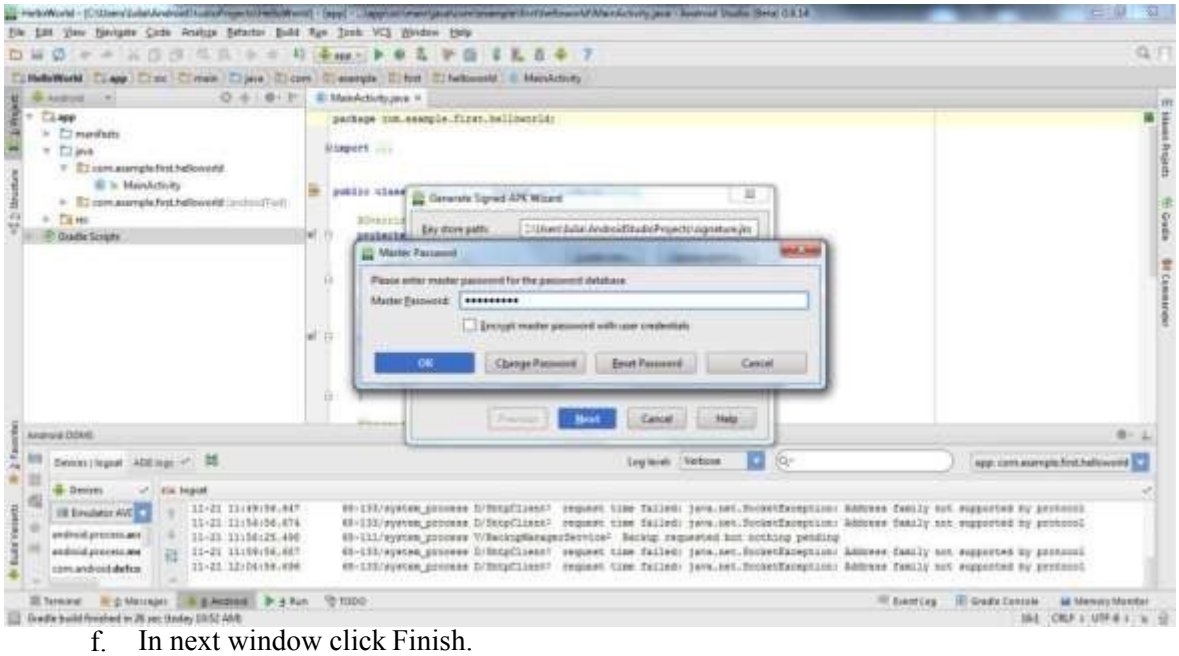


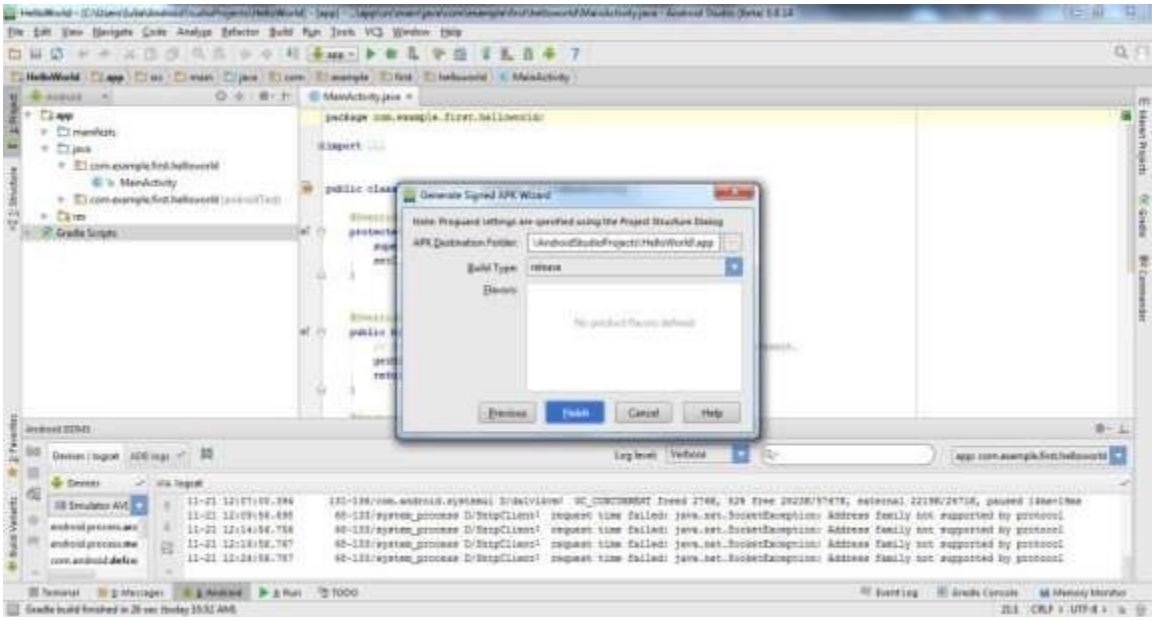
d. Complete the fields. On Key store path you choose a path and a name for your jks (we choose signature for this example). On Password you choose your own password. Click OK.



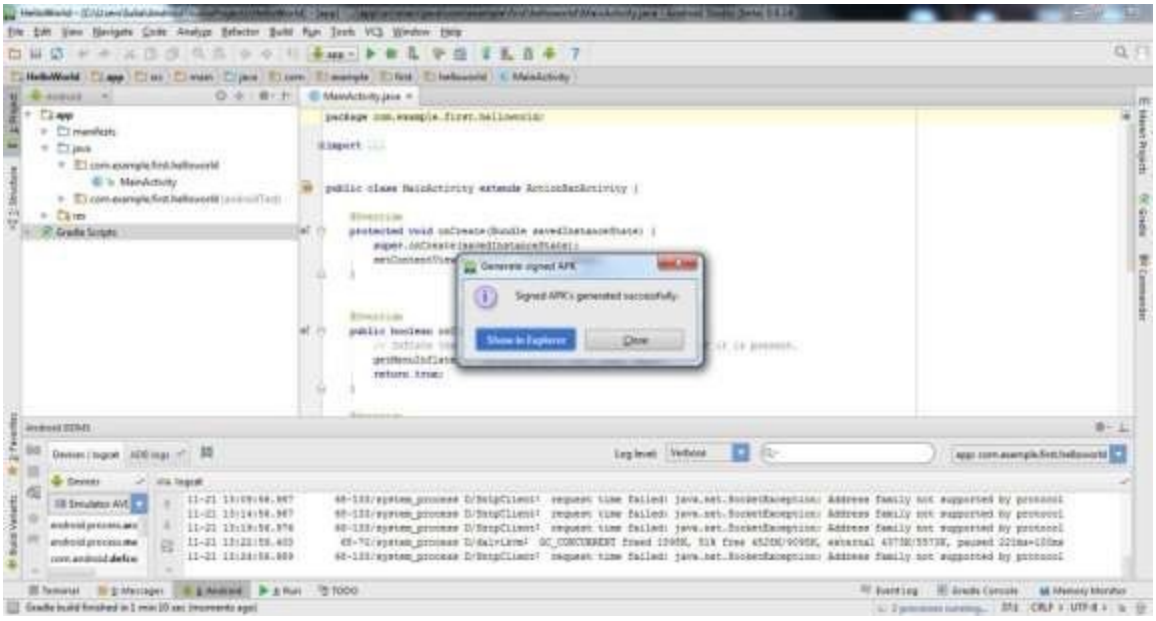
e. In next window rewrite password and click OK.



f. In next window click Finish.



g. You find your app in project folder ->HelloWorld -> app -> app-release.apk.

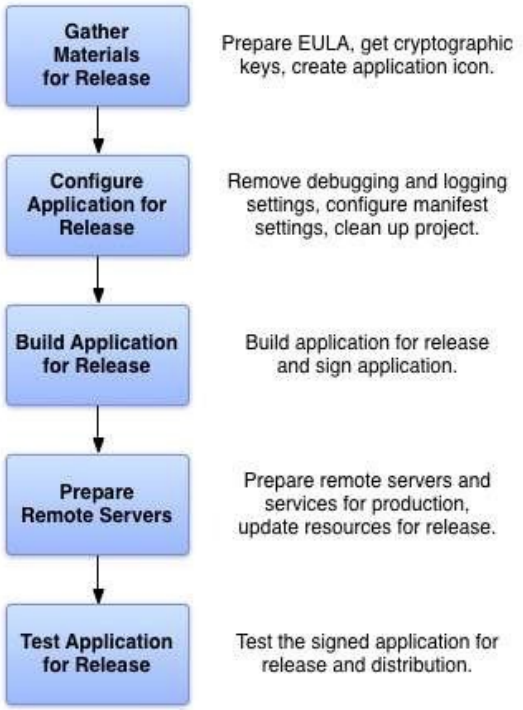


h. You can rename your apk application, copy in mobile device and install it.

## 4. App Publishing Guidance

To release your application to users you need to create a release-ready package that users can install and run on their Android-powered devices. The release-ready package contains the same components as the debug .apk file — compiled source code, resources, manifest file, and so on — and it is built using the same build tools. However, unlike the debug.apk file, the release- ready .apk file is signed with your own certificate and it is optimized with the zipalign tool.

You perform five main tasks to prepare your application for release. The signing and optimization tasks are usually seamless if you are building your application with Android Studio. For example, you can use Android Studio with the Gradle build files to compile, sign, and optimize your application all at once. You can also configure the Gradle build files to do the same when you build from the command line. For more details about using the Gradle build files, see the Build System guide.




To prepare your application for release you typically perform five main tasks (see figure 2). Each main task may include one or more smaller tasks depending on how you are releasing your application. For example, if you are releasing your application through Google Play you may want t while you are configuring your application for release. Similarly, to meet Google Play publishing guidelines you may have to prepare screenshots and create promotional text while you are gathering materials for release.

You usually perform the tasks listed in figure 2 after you have throroughly debugged and tested your application. The Android SDK contains several tools to help you test and debug your Android applications.

**Step 1 Gathering Materials and Resources**

To begin preparing your application for release you need to gather several supporting items. At a minimum this includes cryptographic keys for signing your application and an application icon. You might also want to include an end-user license agreement.

Cryptographic keys

The Android system requires that each installed application be digitally signed with a certificate that is owned by the application's developer (that is, a certificate for which the developer holds the private key). The Android system uses the certificate as a means of identifying the author of an application and establishing trust relationships between applications. The certificate that you use for signing does not need to be signed by a certificate authority; the Android system allows you to sign your applications with a self-signed certificate.

Important: Your application must be signed with a cryptographic key whose validity period ends after 22 October 2033.

You may also have to obtain other release keys if your application accesses a service or uses a third-party library that requires you to use a key that is based on your private key. For example,

if your application uses the MapView class, which is part of the Google Maps external library, you will need to register your application with the Google Maps service and obtain a Maps API key.

Application Icon

Be sure you have an application icon and that it meets the recommended icon guidelines. Your application's icon helps users identify your application on a device's Home screen and in the Launcher window. It also appears in Manage Applications, My Downloads, and elsewhere. In addition, publishing services such as Google Play display your icon to users.

Note: If you are releasing your application on Google Play, you need to create a high resolution version of your icon. See Graphic Assets for your Application for more information.

End-user License Agreement

Consider preparing an End User License Agreement (EULA) for your application. A EULA can help protect your person, organization, and intellectual property, and we recommend that you provide one with your application.

Miscellaneous Materials

You might also have to prepare promotional and marketing materials to publicize your application. For example, if you are releasing your application on Google Play you will need to prepare some promotional text and you will need to create screenshots of your application.

**Step 2 Configuring Your Application for Release**

After you gather all of your supporting materials you can start configuring your application for release. This section provides a summary of the configuration changes we recommend that you make to your source code, resource files, and application manifest prior to releasing your application. Although most of the configuration changes listed in this section are optional, they are considered good coding practices and we encourage you to implement them. In some cases, you may have already made these configuration changes as part of your development process.

Choose a good package name

Make sure you choose a package name that is suitable over the life of your application. You cannot change the package name after you distribute your application to users. You can set the package name in application's manifest file. For more information, see the package attribute documentation.

Turn off logging and debugging

Make sure you deactivate logging and disable the debugging option before you build your application for release. You can deactivate logging by removing calls to Log methods in your source files. You can disable debugging by removing the android:debuggable attribute from the <application> tag in your manifest file, or by setting the android:debuggable attribute to false in your manifest file. Also, remove any log files or static test files that were created in your project.

Also, you should remove all Debug tracing calls that you added to your code, such as startMethodTracing() andstopMethodTracing() method calls.

Important: Ensure that you disable debugging for your app if using WebView to display paid for content or if using JavaScript interfaces, since debugging allows users to inject scripts and extract content using Chrome DevTools. To disable debugging, use the WebView.setWebContentsDebuggingEnabled() method.

Clean up your project directories

Clean up your project and make sure it conforms to the directory structure described in Android Projects. Leaving stray or orphaned files in your project can prevent your application from compiling and cause your application to behave unpredictably. At a minimum you should do the following cleanup tasks:

- Review the contents of your jni/, lib/, and src/ directories. The jni/ directory should contain only source files associated with the Android NDK, such as .c, .cpp, .h, and .mk files. The lib/ directory should contain only third-party library files or private library files, including prebuilt shared and static libraries (for example, .so files). Thesrc/ directory should contain only the source files for your application (.java and .aidl files). The src/ directory should not contain any .jar files.
- Check your project for private or proprietary data files that your application does not use and remove them. For example, look in your project's res/ directory for old drawable files, layout files, and values files that you are no longer using and delete them.
- Check your lib/ directory for test libraries and remove them if they are no longer being used by your application.
- Review the contents of your assets/ directory and your res/raw/ directory for raw asset files and static files that you need to update or remove prior to release.

Review and update your manifest and Gradle build settings

Verify that the following manifest and build files items are set correctly:

2. <uses-permission> element

   You should specify only those permissions that are relevant and required for your application.

   - android:icon and android:label attributes

   You must specify values for these attributes, which are located in the <application> element.

   - android:versionCode and android:versionName attributes.

   We recommend that you specify values for these attributes, which are located in the <manifest> element. For more information see Versioning your Application.

There are several additional manifest or build file elements that you can set if you are releasing your application on Google Play. For example, the android:minSdkVersion and android:targetSdkVersion attributes, which are located in the<uses-sdk> element. For more information about these and other Google Play settings, see Filters on Google Play.

Address compatibility issues

Android provides several tools and techniques to make your application compatible with a wide range of devices. To make your application available to the largest number of users, consider doing the following:

- **Add support for multiple screen configurations**

Make sure you meet the best practices for supporting multiple screens. By supporting multiple screen configurations you can create an application that functions properly and looks good on any of the screen sizes supported by Android.

- **Optimize your application for Android tablet devices.**

If your application is designed for devices older than Android 3.0, make it compatible with Android 3.0 devices by following the guidelines and best practices described in Optimizing Apps for Android 3.0.

- **Consider using the Support Library**

If your application is designed for devices running Android 3.x, make your application compatible with older versions of Android by adding the Support Library to your application project. The Support Library provides static support libraries that you can add to your Android application, which enables you to use APIs that are either not available on older platform versions or use utility APIs that are not part of the framework APIs.

Update URLs for servers and services

If your application accesses remote servers or services, make sure you are using the production URL or path for the server or service and not a test URL or path.

Implement Licensing (if you are releasing on Google Play)

If you are releasing a paid application through Google Play, consider adding support for Google Play Licensing. Licensing lets you control access to your application based on whether the current user has purchased it. Using Google Play Licensing is optional even if you are releasing your app through Google Play.

**Step 3 Building Your Application for Release**

After you finish configuring your application you can build it into a release-ready .apk file that is signed and optimized. The JDK includes the tools for signing the .apk file (Keytool and Jarsigner); the Android SDK includes the tools for compiling and optimizing the .apk file. If you are using Android Studio or you are using the Gradle build system from the command line, you can automate the entire build process.

Building with Android Studio

You can use the Gradle build system, integrated with Android Studio to build a release-ready .apk file that is signed with your private key and optimized. The build process assumes that you have a certificate and private key suitable for signing your application. If you do not have a suitable certificate and private key, Android Studio can help you generate one.

**Step 4 Preparing External Servers and Resources**

If your application relies on a remote server, make sure the server is secure and that it is configured for production use. This is particularly important if you are implementing in-app billing in your application and you are performing the signature verification step on a remote server.

Also, if your application fetches content from a remote server or a real-time service (such as a content feed), be sure the content you are providing is up to date and production-ready.

**Step 5 Testing Your Application for Release**

Testing the release version of your application helps ensure that your application runs properly under realistic device and network conditions. Ideally, you should test your application on at least one handset-sized device and one tablet-sized device to verify that your user interface elements are sized correctly and that your application's performance and battery efficiency are acceptable.

## 2.5. Tips for Launching an App

Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this topic will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application

| Step | Activity |
|---|---|
| 1 | **Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets. |
| 2 | **Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity. |
| 3 | **Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting |

| | |
|---|---|
| | up time zone, localization or any other specific requirement as per the targeted region. |
| 4 | **Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices. |
| 5 | **SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target. |
| 6 | **Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies. |
| 7 | **Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere. |
| 8 | **Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: **Preparing for Release**. |
| 9 | **Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide. |

# 14. Intel XDK

## Objectives

After completing this chapter we will be able to understand the following topics.

- Getting Started with Intel XDK
- Edit Your App
- Testing and Debugging
- Package Options
- Building Your App

## 1. Getting Started with Intel XDK

### a) Install the Intel XDK on Your Development Machine

If you do not already have access to the Intel XDK on your development machine, download it from http://xdk.intel.com.

### b) Install App Preview on Your Test Mobile Device

App Preview lets you run Intel XDK project files on real physical hardware without performing a full build.
If you do not already have the App Preview tool installed on your test mobile device, download this free Intel app from the Apple* App Store, Google* Play Store, or Windows* Phone Store.

### c) Access the Intel XDK

1. Launch the Intel XDK on your development machine.
2. If necessary:
   - Log in.
   - Sign up for an account. User Name = Your email address.

#### Creating a First App

A typical Intel XDK project corresponds to an app and provides a container for your project files and associated settings. There are three main project types:

**Standard HTML5** - This project code base uses HTML5 and JavaScript* code, may use W3C* standard APIs, but does not support the use of Apache Cordova* APIs (aka Cordova* plug-ins). It can be deployed as a packaged hybrid mobile app on many platforms (Android*, Android Crosswalk*, Apple iOS* and Microsoft Windows 8*) or as a hosted web app for certain web- or OS-based platforms.

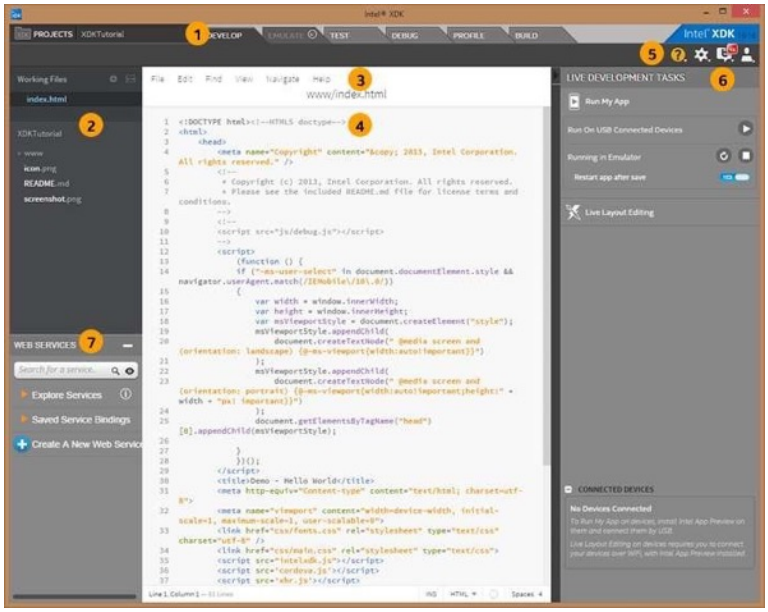
**HTML5 + Cordova** - This project code base uses HTML5 and JavaScript* code, and supports the use of Apache Cordova* APIs (via Cordova* plug-ins). It can be built and deployed as a packaged hybrid mobile app for Android*, Android Crosswalk*, Apple iOS*, and Windows 8* devices.

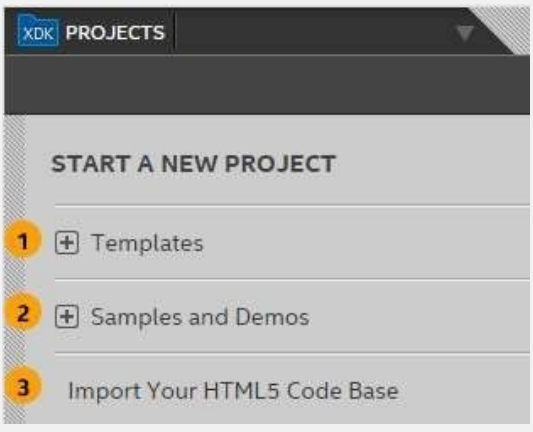
I**nternet of Things embedded apps** (Intel® XDK IoT Edition only) - This project code base consists of embedded JavaScript* intended for execution on a Node.js runtime. This project type is only recognized by the Intel XDK IoT Edition. It requires an IoT maker board and is not built like mobile web apps for phones and tablets.

Let’s jump-start this tutorial using a working, HTML5+Cordova demo app as the base for your new app.
If this is your first project in the Intel XDK - The Intel XDK displays the START A NEW PROJECTpalette in the PROJECTS tab and offers an opportunity to take a Quick Tour when project setup is complete.

You already have active projects in the Intel XDK - Click the PROJECTS tab drop-down arrow to display the YOUR INTEL® XDK PROJECTS dashboard. Then click the START A NEW PROJECTbutton to display the START A NEW PROJECT palette.

**Get to Know the START A NEW PROJECT Palette on the PROJECTS Tab**






| | |
|---|---|
| 1 | **Templates** - Use to build an app based on different types of app user interfaces. You can also choose a simple blank template to start with an essentially empty project. |
| 2 | **Samples and Demos** - Use to open a complete, working project to jump-start your own app, learn how HTML5 apps work, or learn how the Intel XDK works. (This is what we will do in this tutorial.) |
| 3 | **Import Your HTML5 Code Base** - Use to create an Intel XDK project by importing an existing HTML5 app, such as an app created using Adobe PhoneGap* or Apache Cordova* APIs, HTML5 APIs, Intel XDK device APIs, or the appMobi* XDK. |

### Choose the Hello World Demo App

1. On the START A NEW PROJECT palette on the PROJECTS tab, expand the Samples and Demos option to display a General library of Standard HTML5 demos.
2. Above the library area, click HTML5 + Cordova to display a General library of HTML5 + Cordova demos.
3. Scroll to and click the *Hello World* demo image.
4. Click the Continue button to start setting up your new project.

### Create the XDKTutorial Project

1. Name your project XDKTutorial, save it in C:\Users\[userid]\XDKTutorial, and click theCreate button to create your project.
2. On the Congratulations! dialog box, click the LET'S DO IT button to take a Quick Tour that provides a basic explanation of the Intel XDK UI.
3. When the Quick Tour is complete, the Intel XDK displays the DEVELOP tab, which is where you perform most of your app development.
4. Preview Your App on Virtual Devices
5. Because we started from a demo, we can immediately preview the XDKTutorial app in browsers, on real devices, or on virtual devices using the LIVE DEVELOPMENT TASKS palette in the DEVELOP tab.

6. In the LIVE DEVELOPMENT TASKS palette in the **DEVELOP** tab, click **Live Layout Editing** to display preview options similar to the following.


1. Click **Run My App** to display preview options similar to the following.

2. Click the **Run in Emulator** icon to display an **Intel XDK Emulator** floating window similar to the following.

### Get to Know the Intel XDK Emulator Floating Window


| Sl NO | Description |
|---|---|
| 1 | Use the **Intel XDK Emulator** floating window to quickly identify and fix defects before you test your app on an actual mobile device. You can simulate app functionality on a variety of virtual devices using this device simulator based on the Apache Ripple* emulator. You can also launch a built-in version of the CDT debugger from this floating window to debug app functionality. (We will do this later in this tutorial.) |
| 2 | Use the toolbar buttons to reload your app source files and restart your app, launch the built-in debugger in another floating window, display and change emulator settings, and stop executing the app in the emulator. |
| 3 | Use the **Auto Zoom** icon to zoom the device visual representation to the maximum size that fully fits in the window, and the slider to manually resize the device visual representation. |
| 4 | Use the palettes in the accordion-style columns to configure various virtual devices that help you quickly determine how well your app works in different orientations and a variety of screen sizes and aspect ratios. You can open and close each palette, hide and show the columns, move columns, and move palettes within a column. |
| 5 | Use the device visual representation view and test your app. |

### Play with Your App, Virtual Device Skins, and Palettes

- Running your *XDKTutorial* app - How can you possibly resist clicking the Beep! button?
- Choosing a different virtual device - Click the DEVICES palette drop-down arrow and choose among the *skins*.

- Changing virtual device orientation - Click the orientation icons in the DEVICES palette.
- Opening and closing each palette - Click the palette header bar.
- Hiding and showing each palette column - Click the  and  controls.
- Moving a palette - Click and drag a palette header bar to move it up or down the palette column or to another column.

More Handy Information

- The emulator is also accessible from the EMULATE tab.
- The Intel XDK Emulator floating window is also accessible by clicking the Undock icon at the top right corner of the EMULATE tab label.
- The Undock icon at the top right corner of the EMULATE tab label changes to a Dock icon and the EMULATE tab is disabled when the Intel XDK Emulator floating window is displayed. To re-enable the EMULATE tab, either close the floating window or click the Dock icon at the top right corner of the EMULATE tab label.
- The emulator is actually a web app that runs inside a node webkit. Your app runs within an inner HTML frame.
- The runtime engine rendering your HTML5 code in the emulator is based on the Chromium open source browser. This up-to-date web runtime engine may implement HTML5 features more correctly than the web runtime on a real mobile device, especially if that mobile device has an old OS version.
- Previous versions of the Intel XDK or appMobi* software refer to appmobi.js in source files. Replace all mentions of appmobi.js with intelxdk.js.
- The emulator does not support the use of *.mp3 audio files.

## 2. Edit Your App

Let's add some functionality to your *XDKTutorial* app. If necessary, click the **DEVELOP** tab to display a window similar to the following.

### Get to Know the DEVELOP Tab

| Sl | Descriptio |
|---|---|
| 1 | Use the **DEVELOP** tab to edit files and add files to your project. You can use the built-in code editor or your favorite code editor. |
| 2 | Use the file tree to view the files associated with the active project, including image files, and to choose a file for editing. |
| 3 | Use the menu options (or equivalent shortcut keys) to perform common code editor functions. |
| 4 | Use the code editor view to edit file contents. Context menus are also available. |
| 5 | Use the Intel XDK toolbar buttons (present on all tabs) to access Help information for the displayed tab and general Help information, start the Quick Tour, display and change Intel XDK settings, view recent Twitter Tweets* about the Intel XDK, and display and change account settings. |
| 6 | Use the **LIVE DEVELOPMENT TASKS** palette to preview your code in a browser, on a real mobile device, or on virtual devices. Notice the *XDKTutorial* app is still running in the **Intel XDK Emulator**floating window and is set to restart after you save your changes to project files. With **Live Editing Layout**, code changes appear immediately after you make your edits using the built-in code editor, or after you save project files using an external editor. |
| 7 | Use the **WEB SERVICES** palette to explore built-in third-party web service APIs (cloud services) as well as integrate other third-party web service APIs. This tutorial does not cover web services. See theDevelop Overview for more information. |

### Use the Built-in Code Editor to Edit Your Code

1. Use the built-in code editor to uncomment <script src="js/debug.js"></script> near line 10.
2. Choose File > Save to save your edit and change the contents of the Intel XDK Emulator floating window.

More Handy Information

- The built-in code editor is based on the Brackets* editor, so you can easily add Brackets* extensions (File > Extension Manager).
- If you use a code editor external to the Intel XDK, you must click the Reload App icon on the toolbar to update all files and restart your app when you return to the emulator.

- If you create an app using the App Designer (*Hello World* demo files were not created with the App Designer), the Intel XDK offers the CODE and DESIGN views in the DEVELOP tab so you can switch back and forth between the built-in code editor and the drag-and-drop HTML5 layout editor
- The App Designer editor is available only when you are editing HTML5 files created with the editor.
- The App Designer is a *round-trip* editor. For example, you can start creating your apps in the App Designer, switch to the code editor to view the code corresponding to your layout efforts, modify code in the code editor, then switch back to the App Designer to view those modifications – as long as you do not change the App Designer class="uib*" tag labels
(*uib* stands for *user interface builder*).
- Any elements the App Designer does not recognize are still rendered in the **DESIGN** view; however, they are not editable in the **DESIGN** view.

## 3. Testing and Debugging

**Test App Functionality**

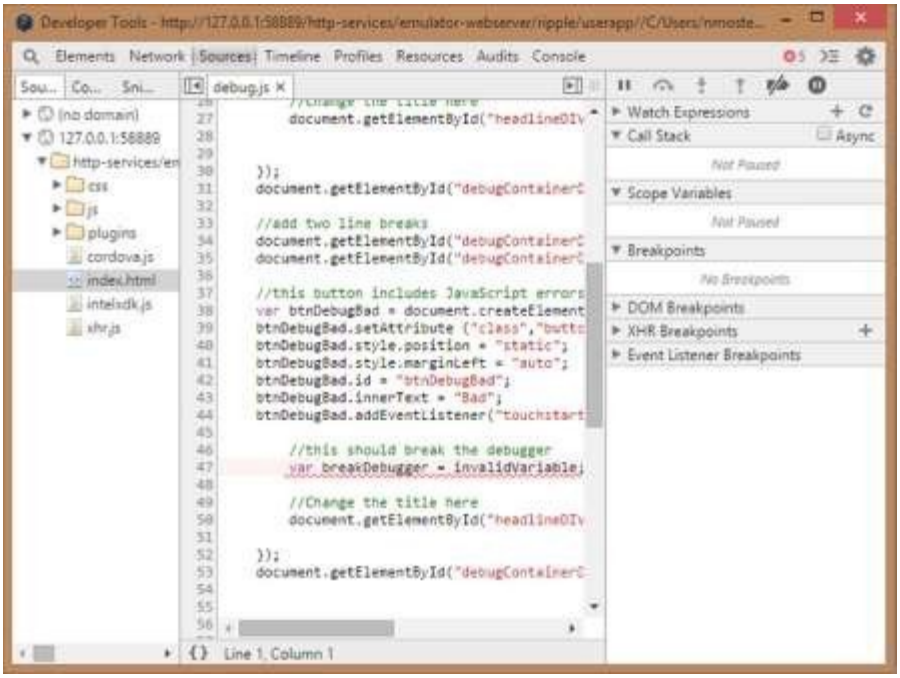
**Caution**: The processor on your development machine is probably faster than the processor on a real mobile device, so performance problems are typically not visible in the emulator.

1. Notice there are two new buttons in your *XDKTutorial* app in the Intel XDK Emulator floating window: Good and Bad.
2. Click the Good button. Notice the header changes to Hello Good!
3. What do you think will happen when you click the Bad button? Try it. Not what you expected? Guess it is time to debug.

### Use the Built-in CDT Debugger to Debug Your App

1. Click the  Launch Debugger icon on the Intel XDK Emulator toolbar to open a built-in version of the CDT debugger in a floating window.
2. Click the Sources button to open a file.
3. Press Ctrl+O and choose the debug.js file.

4. Scroll down to near line 46 to display a window similar to the following.




There is the problem!

### Correct App Code

Caution: Modifying code in the debugger window impacts future app behavior in the emulator; however it does not modify actual source code, so make your source code modifications in the DEVELOP tab.

1. Close the Developer Tools floating window.
2. If necessary, click the DEVELOP tab.
3. In the file tree, open the www/js folder and choose the debug.js file.
4. Scroll to near line 46 and comment out var breakDebugger = invalidVariable;>
5. Choose **File** > **Save**.

### Retest App Functionality

Now try clicking the **Bad** button in the **Intel XDK Emulator** floating window. Better? If not, click the the  **Reload App** icon on the toolbar and try again.

### More Handy Information

- What you are actually debugging is a simulation of a real mobile device using the Chromium desktop engine augmented with Cordova* and Intel XDK APIs. This simulation is designed to provide an idea of how your app will render on various devices and form factors. Some visual aspects of your app may render differently on real devices, especially if the real devices have an old OS version.
- Using the built-in version of the CDT debugger, you can set breakpoints, single-step, display variables in your JavaScript* code, do full DOM debugging, and see the effects of CSS on the DOM. You also have access to the CDT JavaScript* console, where you can view your app console.log messages and interact with your app JavaScript* context by manually inspecting properties and executing methods.

### Run Your App on a Real Mobile Device

Now that you are confident your *XDKTutorial* app works properly on virtual devices, it is time to run it on a real mobile device.

1. Click the **TEST** tab.
2. Click the **I HAVE INSTALLED APP PREVIEW** button - because you already did this, right? ;-)
3. Your development machine displays a **Please sync with our testing server** message the first time you click the **TEST** tab for a project. Click the **DISMISS** button so we can first explore the**TEST** tab.

### Get to Know the TEST Tab

| Sl No | Descriptio |
|---|---|
| 1 | Use the **TEST** tab to evaluate - over the network - how your app looks and performs on real physical hardware without performing a full build. |

| | |
|---|---|
| **2** | Use the **MOBILE** button to test via a test server in the cloud. Advantages: Requires minimal setup; avoids potential firewall and network topography issues; you can pull your files down anytime from anywhere in the cloud. Use the **WIFI** button to test via a local Wi-Fi network to which both your development machine and test mobile device are connected. Advantages: Usually faster once set up; does not consume mobile data; pulls files directly from your development machine. |
| **3** | Use the instructions as a refresher for testing via mobile or Wi-Fi, with or without the App Preview QR code reader. |
| **4** | Use the **PUSH FILES** button to push the most recent project files on your development machine to the test server in the cloud. |
| **5** | Use the QR code with the App Preview QR code reader (accessible from the camera icon) to launch your app on your test mobile device. **Tip**: Alternatively, you can launch your app from the **Server Apps** list (**MOBILE** mode) or **Local Apps**list (**WIFI** mode) on the App Preview tool. |
| **6** | You can also remotely debug your app while it runs on real physical hardware using the **TEST** tab and the weinre* (which stands for *web inspector remote*) debug console. |

### More Handy Information

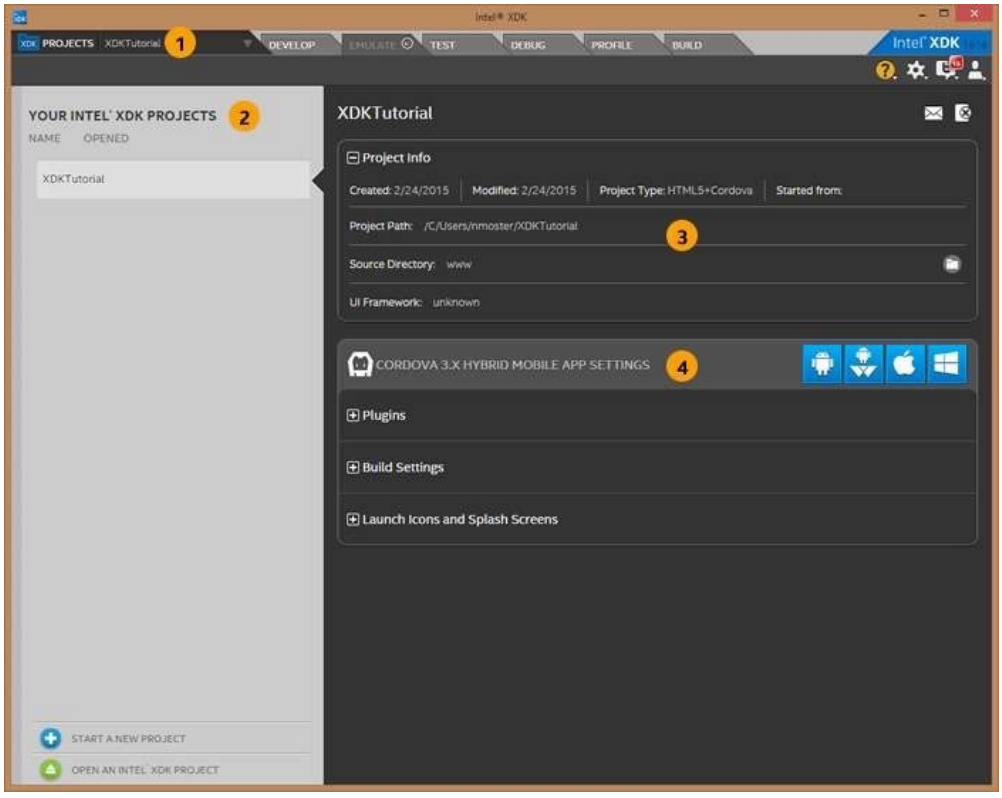
App Preview keeps track of all the apps you upload to the test server.
What you are actually testing on the real physical hardware is your app loaded into a platform-specific container app that is representative of the native wrapper included with your app when you build your app.

Android* mobile devices: If you cannot access the Google* Play Store, click the **Information** icon to email the App Preview download to your mobile device or copy the App Preview download link to your clipboard.

## 4. Package Options

### Check Your App Package Options

Now that you are confident the XDKTutorial app works properly on a real mobile device, it is time to build your app. But let's check your app package options first. Click the PROJECTS tab to display a window similar to the following.



## Get to Know the PROJECTS Tab

| Sl No | Description |
|---|---|
| 1 | Use the **PROJECTS** tab to manage projects and project information, including options to package your app for submission to app stores. |
| 2 | The **YOUR INTEL® XDK PROJECTS** dashboard lists all known Intel XDK projects on your development machine.<br>The Intel XDK operates on only one project at a time. The currently active project is always marked in the **YOUR INTEL® XDK PROJECTS** dashboard and is the project on which all other tabs operate. |
| 3 | All your project files are stored locally on your development machine.<br>The **Source Directory** is the root location of your sources, including the index.html file, which is the main entry point to your project. The **Source Directory** and the **Project Path** may or may not refer to the same directory location, but the root **Source Directory** must be at or below the **Project Path**. |
| 4 | **NOTE**: If you plan to build your app using Cordova* 2.x APIs instead of Cordova* 3.x APIs, specify package options in the **BUILD** tab instead of the **PROJECTS** tab.<br>Use **CORDOVA 3.X HYBRID MOBILE APP SETTINGS** to generate platform-specific intelxdk.conf.<platform>.xml configuration files when you upload your app to the build server for packaging. These configuration files are then stored in your project directory.<br>Plug-ins provide a way to extend your app JavaScript* APIs, resulting in a tighter integration between your app and mobile device software and hardware.<br>Use **PLUGINS AND PERMISSIONS** to specify standard Cordova* 3.x plug-ins, featured and custom Cordova* plug-ins, Intel XDK plug-ins, additional third-party plug-ins, and platform-specific permissions in addition to permissions required by chosen plug-ins. Use **BUILD SETTINGS** to specify details typically needed by app stores. Use **LAUNCH ICONS AND SPLASH SCREENS** to choose orientation, icons, and splash screens. See the Projects Overview for more information. |

### Explore Your App Package Options

Take a moment to check out your app package options. When you are ready to submit your own apps to various app stores, you will need to supply values, such as App ID, App Name, and App Description. The default values are sufficient for this tutorial.

### More Handy Information

- Each project has a <project-name>.xdk file in the root directory. This file does not contain any application code and is not required for your app to execute on a real mobile device.
- Deleting a project from the project list does not delete the project files or directory from your development machine. You can later reopen your project and add it back to your project list using the OPEN AN INTEL® XDK PROJECT button on the YOUR INTEL® XDK PROJECTS dashboard on the PROJECTS tab.
- The emulator notes the core Cordova* plug-ins chosen on the PROJECTS tab and presents to your app the APIs corresponding to those core Cordova* plug-ins when your app runs inside the emulator. The emulator presents to your app the complete set of APIs provided by the Intel XDK plug-ins, regardless of what is chosen on the PROJECTS tab.
- At this time, only the BUILD tab makes use of third-party plug-ins specified in the PROJECTS tab.

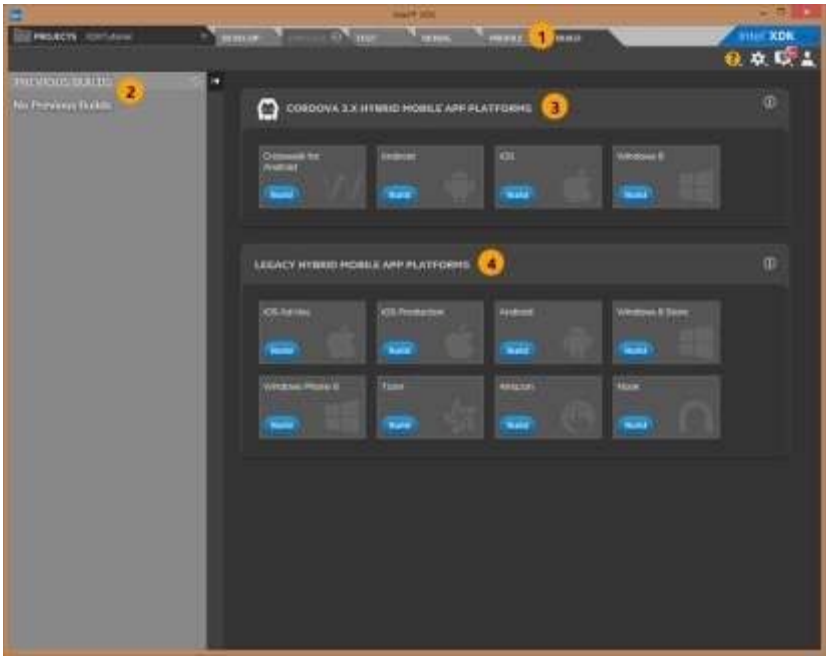
## 5. Build Your App

It is time to build your app! Click the BUILD tab to display a window similar to the following.

Get to Know the BUILD Tab.

| Sl NO | Description |
|---|---|
| 1 | Use the **BUILD** tab to:<br><br>Package your app and deploy it as locally installed hybrid mobile app, thereby enabling the use of the same distribution and monetization channels, as well as the same app download, installation, and launch experience as native mobile apps.<br><br>Create an HTML5 bundle you can submit to certain app stores (such as the Google Chrome* Web Store) or place (hosted) on web servers. |
| 2 | Use the **PREVIOUS BUILDS** palette to view (and refresh) a list of previous builds. You can also hide/show the entire **PREVIOUS BUILDS** palette. |

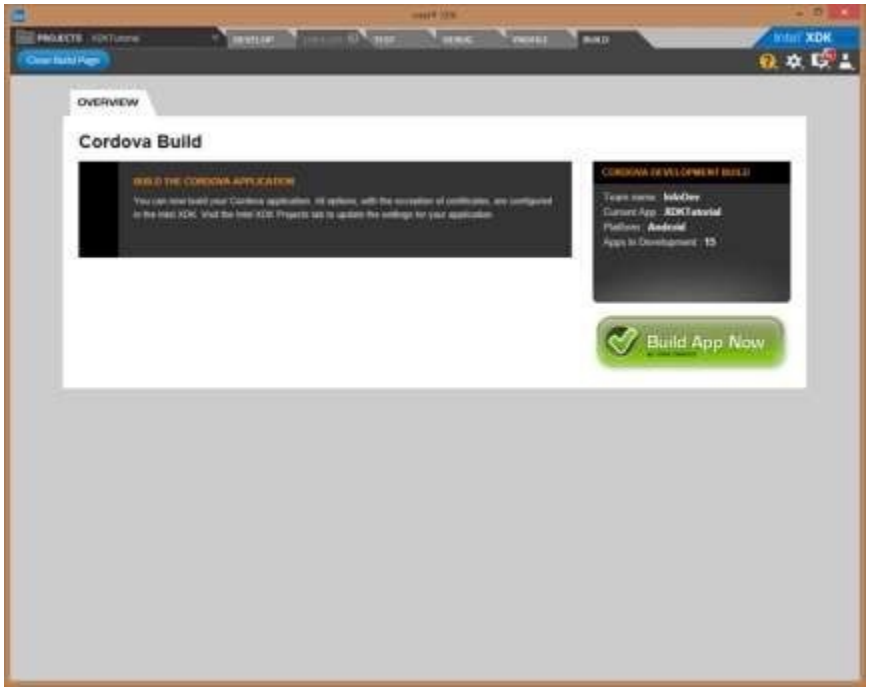
| | |
|---|---|
| **3** | Use **CORDOVA 3.X HYBRID MOBILE APP PLATFORMS** to create a native app package suitable for submission to app stores. Packages support Cordova* 3.x core APIs and selected plug-ins, and Intel XDK APIs. See the Projects Overview for more information.<br><br>Crosswalk is an HTML5 application runtime based on a variety of open source projects. The Intel XDK distribution of the Crosswalk runtime also supports many cutting-edge APIs under consideration for standardization by the W3C* organization. See Crosswalk Overview for more information. |
| **4** | Use **LEGACY HYBRID MOBILE APP PLATFORMS** to create a native app package suitable for submission to app stores. Packages support Cordova* 2.x APIs and Intel XDK APIs. See the Build Tab Overview for more information.<br><br>The Tizen* OS is an open source, standards-based, software platform for devices such as smart phones, tablets, netbooks, in-vehicle infotainment devices, and smart TVs. See the Build Tab Overview for more information. |




NOTE: If your app is a Standard HTML5 project type instead of an HTML5 + Cordova project type, the Build tab also includes a BUILD AS A WEB APP region.

### Package Your App as a Mobile App for an Android* Store

1. Click the Android BUILD button in the CORDOVA 3.X HYBRID MOBILE APP PLATFORMSregion to connect to the build server, upload your XDKTutorial app to the cloud, and display a window similar to the following.




2. Notice all packaging configuration has already been performed (in the PROJECTS tab); there is nothing preventing you from building your XDKTutorial app right now. Click the Build App Nowbutton to build the XDKTutorial app as an Android* mobile app and ultimately display a window similar to the following.

If the build fails, click the link to review a more detailed buildlog.

The Intel XDK does not provide actual store submission services for your app; however, it does explain a variety of possible next steps (which we obviously will not perform for this *XDKTutorial*app).

Click the **Close Build Page** button.

### More Handy Information

- Builds are performed in the cloud, so you do not need to install and configure an SDK or native development system for each target platform (as you must do if you are building a Cordova*/ PhoneGap* app).
- You must obtain the proper developer credentials to submit apps to most appstores.
- With the exception of Crosswalk for Android* packages, all packages use the built-in webview (embedded browser) that is part of the target mobile device firmware to execute (render) your app. For example, Android* packages use the Android* browser webview built into the Android* mobile device, and iOS* packages use the Apple Safari* browser webview built into the iOS* mobile device.