

CALCULATOR APP - ANDROID APP DEVELOPMENT

Submitted in partial fulfillment of the requirements for the award
of Bachelor degree in Computer Science

By

S.Kirupalakshmi (Reg.No : 38290039)

D.Blessy (Reg.No : 38290011)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

**SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU.**

April 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(Established under Section 3 of UGC Act, 1956)

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai- 600 119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of S.Kirupalakshmi (Reg.No : 38290039) and D.Blessy (Reg.No : 38290011), who carried out the project entitled “CALCULATOR APP - ANDROID APP DEVELOPMENT ” under our supervision from November 2020 to April 2021.

Internal Guide

Dr.J.ALBERT MAYAN M.E., Ph.D.)

Head of the Department

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

DECLARATION

We S.Kirupalakshmi (Reg.No : 38290039) and D.Blessy (Reg.No : 38290011) hereby declare that the project Report on “ANDROID APP DEVELOPMENT” done by me under the guidance of Dr.J.ALBERT MAYAN M.E., Ph.D , is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science .

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., PH.D., Dean**, School of Computing **Dr.S. Vigneshwari M.E., Ph.D** and **Dr. L. Lakshmanan M.E., Ph.D., Heads of the Department**, for providing me necessary Support and details at the right time during the progressive reviews..

I would also like to express our sincere thanks to my internal guide **Mr.J.ALBERT MAYAN M.E.,(Ph.D.,)** Assistant Professor guided me in the preparation of the report. Also, I thank the **Almighty** and my **Parents** for supporting me in the completion of the professional Training

ABSTRACT

Starting with the basics of the Android application development, it can run by using the Java development kit which plays an important role in creating any application regarding to the Android. At present, we are developing the CALCULATOR APP by using the android development studio. While creating this app first we need a sample activity diagram as shown in the screen shots. We need to write the source code regarding the app we are creating. After writing the source code, we need to compile it and the written source code will be in the Java script. The compilation process can be done in the tools options. For building the required app we need to go for the build apk which will be presented in the options.

After building the apk the debug app will be created if your written source code is correct and it will be displayed as build successful. If it is not successful then you should modify the source code and correct the errors. The debug app created cannot run in the system because the server is not available. So, we are inserting this debug app into the android mobile by using the USB cable. And it is stored in the SD card or internal storage as the user required. After doing this, the installation process is being done. And we get the required app we want.

TITLE	CHAPTER NO	PAGE NO
	ABSTRACT	ix
	LIST OF FIGURES	xii
	LIST OF TABLES	xi
1.INTRODUCTION		
	1.1 OUTLINE OF THE PROJECT	1
	1.2 ABOUT ANDROID	2
	1.3 FEATURES IF ANDROID	2
	1.4 ANDROID AND ITS APPLICATIONS	2
2.ARCHITECTURE		
	2.1 ANDROID RUNTIME	4
	2.2 ENVIRONMENT SETUP	5
	2.3 SYSTEM REQUIREMENTS AND ITS INSTALLATONS	7
	2.4 INSTALLATION ON WINDOWS	7
	2.5 ANDROID SETUP AND ITS WIZARD	8
	2.6 INSTALLING THE ANDROID SDK PACKAGES	9
	2.7 CREATING THE ANDROID VIRTUAL DEVICE	10
3.ANDROIDAPPLICATIONS		
	3.1 ADDITIONAL COMPONENTS	14
	3.2 ACCESSING RESOURCES	18

	3.3 OVERVIEW OF APPLICATION	19
	3.4 SERVICES	22
	3.5 CONTENT PROVIDERS	25
	3.6 FRAGMENTS	27
4.SDK TOOLS		
	4.1 PLATFORM TOOLS	32
	4.2 INTERNAL STORAGE	33
5.APPENDIX		
	5.1 SCREEN SHOTS	34
6.SOURCECODE		44
7.REFERENCE		49

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NUMBER
1.1	FEATURES OF THE ANDROID	2
3.0	APPLICATIONS OF ANDROID	14
3.1.	ADDITIONAL COMPONENTS	8
3.1.1	RESOURCES ORGANIZING AND ACCESSING	9
3.3	OVERVIEW OF THE APPLICATIONS	19
3.4	SERVICES	19
3.4.1	CALLBACK AND DESCRIPTION	20
4.0	SDK TOOLS	30

LIST OF FIGURES

FIGURE NUMBER	FIGURE NAME	PAGE NUMBER
2.1	ARCHITECTURE	5
3.1	TYPES OF SERVICES	21
3.2	BROAD CAST SERVICES	23
3.3	CONTENT PROVIDERS	24
3.4	CREATE CONTENT PROVIDERS	25
3.5	FRAGMENTS	27
3.6	LIFE CYCLE OF FRAGMENTS	28

LIST OF ABBREVIATIONS

SYNONYMS	EXPLANATION
ADB	ANDROIDDEBUG BRIDGE
AIDL	ANDROID INTERFACE DEFINITION LANGUAGE
ADK	ANDROID DEVELOPMENT KIT
AVD	ANDROID VIRTUAL DEVICE
DVM	DALVIK VIRTUAL MACHINE
GPS	GLOBAL PROTECTING SYSTEM
IPC	INTER PROCESSOR COMMUNICATION
IPS	INTER PROCESSOR SYSTEM
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
JVM	JAVAVIRTUAL MACHINE
SDK	SOFTWARE DEVELOPMENT KIT

CHAPTER 1

INTRODUCTION

1.1. OUTLINE OF THE PROJECT

Beginning with the basics, this book provides an outline of the steps necessary to set up an Android development and testing environment. An overview of Android Studio is included covering areas such as tool windows, the code editor and the Designer tool. An introduction to the architecture of Android is followed by an in-depth look at the design of Android applications and user interfaces using the Android Studio environment. More advanced topics such as database management, content providers and intents are also covered, as are touch screen handling, gesture recognition, camera access and the playback and recording of both video and audio. This edition of the book also covers printing, transitions and cloud-based file storage.

- [1] In addition to covering general Android development techniques, the book also includes Google Play specific topics such as implementing maps using the Google Maps Android API, in-app billing and submitting apps to the Google Play Developer Console.
- [2] Chapters also cover advanced features of Android Studio such as Gradle build configuration and the implementation of build variants to target multiple Android device types from a single project code base.
- [3] Assuming you already have some Java programming experience, are ready to download Android Studio and the Android SDK, have access to a Windows, Mac or Linux system and ideas for some apps to develop, you are ready to get started.

This project is to be considered to be android app development. It will be more like an static android app with definite html and java used. All the dynamic content that is being loaded in the android app, will be supported by java script frame works.The present process includes the developement of android app where we use ANDROID STUDIO for creating the source code and it willl bee run by the AVD as per the javascript will be done by the JVM(JAVA VIRTUAL MACHINE.

1.2.ABOUT ANDROID

Android is an open-source, Linux-based operating system for mobile devices such as smartphones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies. Android offers a unified approach to application development for mobile devices which means developers need to develop only for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007, whereas the first commercial version, Android 1.0, was released in September 2008. On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

1.3. FEATURES OF ANDROID

Android is a powerful operating system competing with Apple 4GS and support great features. Few of them are listed below in the table no 1.1.

Table NO 1.1. FEATURES OF ANDROID

Feature	Description
Beautiful UI	Android OS basic screen provides a beautiful and intuitive user interface.
Connectivity	GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage	SQL, a lightweight relational database, is used for data storage purposes.

1.4. ANDROIDAPP AND ITS APPLICATIONS

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be packaged easily and sold out either through a store such as Google Play or the Amazon Appstore. Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile

platformed is growing fast. Every day more than 1 million new Android devices are activated worldwide. This tutorial has been written with an aim to teach you how to develop and package Android application. We will start from environment setup for Android application programming and then drill down to consider various aspects of Android applications.

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars, and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics.

Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, along with the founding of the Open Handset Alliance – a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. Beginning with the first commercial Android device in September 2008, the operating system has gone through multiple major releases, with the current version being 7.0 "Nougat", released in August 2016. Android applications ("apps") can be downloaded from the Google Play store, which features over 2.7 million apps as of February 2017. Android has been the best-selling OS on tablets since 2013, and runs on the clear majority^[a] of smartphones. As of May 2017, Android has two billion monthly active users, and it has the largest installed base of any operating system.

CHAPTER 2

ARCHITECTURE

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram. Linux kernel at the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. Also, the kernel handles all the things that Linux is good at, such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware. Libraries On top of Linux kernel there is a set of libraries including open-source Web browser engine Web Kit, well known library, SQLdatabase which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

2.1. Android Runtime

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

The Application Framework layer provides many higher-level services to applications in the form of Java class. Application developers can make use of these services in their applications. Applications You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games, etc. And the diagram is represented in the fig 21

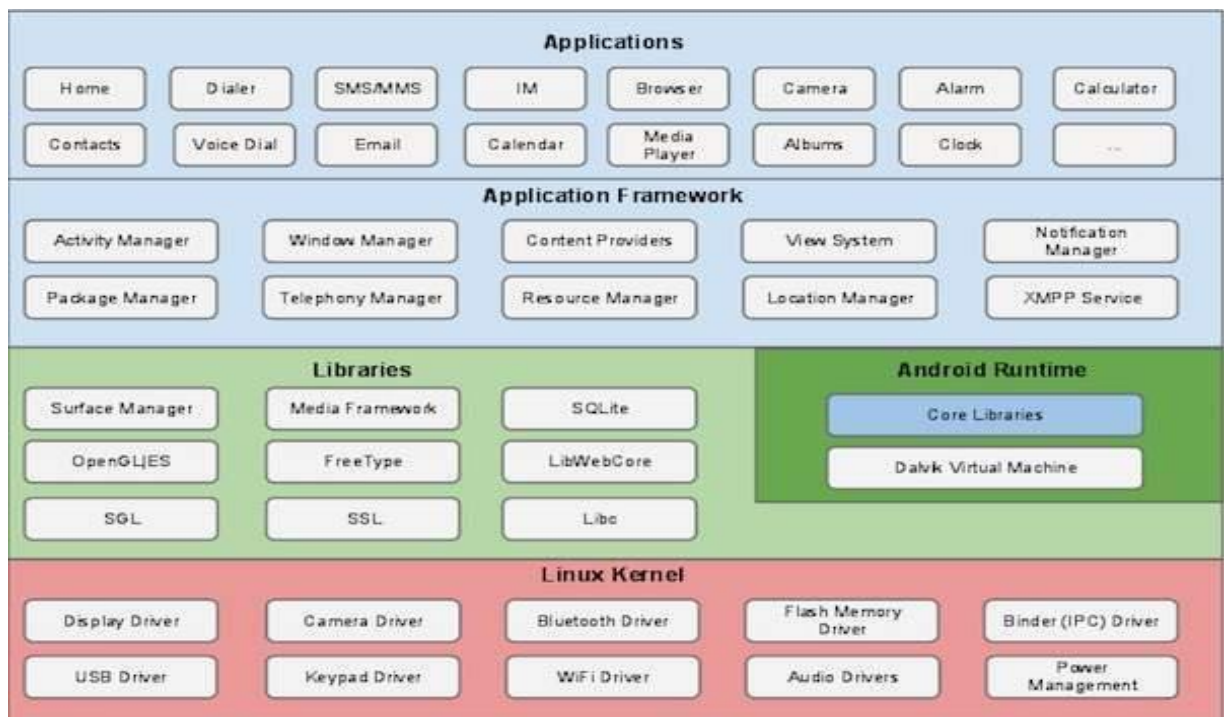


FIG 2.1 ARCHITECTURE

2.2. ENVIRONMENT SETUP

You can download the latest version of Java JDK from Oracle's Java site – Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains **java** and **javac**, typically `java_install_dir/bin` and `java_install_dir` respectively. If you are running Windows and installed the JDK in `C:\jdk1.8.0_102`, you would have to put the following line in your `C:\autoexec.bat` file.

```
set PATH=C:\jdk1.8.0_102\bin;
set JAVA_HOME=C:\jdk1.8.0_102
```

Alternatively, you could also right-click on My computer, select properties, then advanced, then Environment Variables. Then, you would update the PATH value and press the OK button. Second point is that all the required tools to develop Android applications are freely available and can be downloaded from the Web. Following is the list of software's you will need before you start your Android application programming.

- Java JDK5 or later version
- Android Studio

Here last two components are optional and if you are working on Windows machine then these components make your life easy while doing Java based application development. So, let us have a look how to proceed to set required environment. There are so many sophisticated Technologies are available to develop android applications, the familiar technologies, which are predominantly using tools as follows

- Android Studio
- Eclipse IDE(Deprecated)

Before any work can begin on the development of an Android application, the first step is to configure a computer system to act as the development platform. This involves many steps consisting of installing the Java Development Kit (JDK) and the Android Studio Integrated Development Environment (IDE) which also includes the Android Software Development Kit (SDK). This chapter will cover the steps necessary to install the requisite components for Android application development on Windows, Mac OS X and Linux based systems. An activity represents a single screen with a user interface just like window or frame of Java. Android activity is the subclass of Context Them Wrapper class.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram: The Activity class defines the following callbacks i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

2.3.SYSTEM REQUIREMENTS AND ITS INSTALLATION

Android application development may be performed on any of the following system types:

Windows 2003 (32-bit or 64-bit) Windows Vista (32-bit or 64-bit) Windows 7 (32-bit or 64-bit)

Windows 8 / Windows 8.1 Mac OS X 10.8.5 or later (Intel based systems only)

Linux systems with version 2.1or later of GNU C Library (glibc) Minimum of 2GB of RAM (4GB is preferred) 1.5GB of available disk space

The Android SDK was developed using the Java programming language. Similarly, Android applications are also developed using Java. As a result, the Java Development Kit (JDK) is the first component that must be installed. Android development requires the installation of either version 6 or 7 of the Standard Edition of the Java Platform Development Kit. Java is provided in both development (JDK) and runtime (JRE) packages. For the purposes of Android development, the JDK must be installed. If a suitable JDK is not already installed on your system, download the latest JDK package that matches the destination computer system. Once downloaded, launch the installation executable and follow the on-screen instructions to complete the installation process.

If Java is not installed, issuing the “java” command in the terminal window will result in the appearance of a message which reads as follows together with a dialog on the desktop providing a More Info button which, when clicked will display the Oracle Java web page:

No Java runtime present, requesting install. On the Oracle Java web page, locate and download the Java SE 7 JDK installation package for Mac OS X.

Open the downloaded disk image.

2.4. INTSALLATION ON WINDOWS

Locate the downloaded Android Studio installation executable file (named android-studio-bundle-<version>.exe) in a Windows Explorer window and double click on it to start the installation process, clicking the Yes button in the User Account Control dialog if it appears. Once the Android Studio setup wizard appears, work through the various screens to configure the installation to meet your requirements in terms of the file system location into which Android Studio

should be installed and whether it should be made available to other users of the system.

Although there are no strict rules on where Android Studio should be installed on the system, the remainder of this book will assume that the installation was performed into a sub-folder of the user's home directory named `android-studio`. Once the options have been configured, click on the Install button to begin the installation process. On versions of Windows with a Start menu, the newly installed Android Studio can be launched from the entry added to that menu during the installation. On Windows 8, the executable can be pinned to the task bar for easy access by navigating to the `android-studio\bin` directory, right-clicking on the executable and selecting the Pin to Taskbar menu option. Note that the executable is provided in 32-bit (`studio`) and 64-bit (`studio64`) executable versions. If you are running a 32-bit system be sure to use the `studio` executable.

To install the package, simply drag the Android Studio icon and drop it onto the Applications folder. The Android Studio package will then be installed into the Applications folder of the system, a process which will typically take a few minutes to complete. To launch Android Studio, locate the executable in the Applications folder using a Finder window and double click on it. When attempting to launch Android Studio, an error dialog may appear indicating that the JVM cannot be found. If this error occurs, it will be necessary to download and the Mac OS X Java 6 JRE package on the system.

2.5. ANDROID STUDIO SETUP WIZARD

The first time that Android Studio is launched after being installed, a dialog will appear providing the option to import settings from a previous Android Studio version. If you have settings from a previous version and would like to import them into the latest installation, select the appropriate option and location. Alternatively, indicate that you do not need to import any previous settings and click on the OK button to proceed. Click on the Next button, choose the Standard installation option and click on Next once again. On the license agreement screen, select and accept each of the licenses listed before clicking on Finish to complete the setup process. The Welcome to Android Studio screen should appear

From time to time new versions of Android Studio and the Android SDK are released. New versions of the SDK are installed using the Android SDK Manager. Android Studio will typically notify you when an update is ready to be installed. This task can be performed using the Android SDK Manager, which may be launched from within the Android Studio tool.

Click on the Next button, choose the Standard installation option and click on Next once again. On the license agreement screen, select and accept each of the licenses listed before clicking on Finish to complete the setup process. The Welcome to Android Studio screen should then appear.

2.6. INSTALLING THE LATEST ANDROID SDK PACKAGES

The steps performed so far have installed Java, the Android Studio IDE and the current set of default Android SDK packages. Before proceeding, it is worth taking some time to verify which packages are installed and to install any missing packages. This task can be performed using the Android SDK Manager, which may be launched from within the Android Studio tool by selecting the Configure SDK Manager option from within the Android Studio welcome dialog. Once invoked, the SDK Manager tool will appear. Within the Android SDK Manager, make sure that the following packages are listed as Installed in the Status column:

- Tools > Android SDK Platform-tool
- Tools > Android SDK Build-tools
- SDK Platform (most recent version) > SDK Platform
- SDK Platform (most recent version) > ARM EABI v7a System Image
- Extras > Android Support Repository
- Extras > Android Support Library
- Extras > Google Repository
- Extras > Google USB Driver (Required on Windows systems only)
- Extras > Intel x86 Emulator Accelerator (HAXM installer)

In the event that any of the above packages are listed as Not Installed, simply select the checkboxes next to those packages and click on the Install packages button to initiate the installation process. In the resulting dialog, accept the license agreements before clicking on the Install button. The SDK Manager will

then begin to download and install the designated packages. As the installation proceeds, a progress bar will appear at the bottom of the manager window indicating the status of the installation.

2.7. CREATING AN ANDROID VIRTUAL DEVICE(AVD)

During developing Android apps in Android Studio, it will be necessary to compile and run an application multiple times. An Android application may be tested by installing and running it either on a physical device or in an ANDROID VIRTUAL DEVICE(AVD)emulator environment. Before an AVD can be used, it must first be created and configured to match the specification of a device model. The goal of this chapter, therefore, is to work through the steps involved in creating such a virtual device using the Nexus 7 tablet.

AVDs are essentially emulators that allow Android applications to be tested without the necessity to install the application on a physical Android based device. An AVD may be configured to emulate a variety of hardware features including options such as screen size, memory capacity and the presence or otherwise of features such as a camera, GPS navigation support or an accelerometer. As part of the standard Android Studio installation, many emulator templates are installed allowing AVDs to be configured for a range of different devices. Additional templates may be loaded or custom configurations created to match any physical Android device by specifying properties such as processor type, memory capacity and the size and pixel density of the screen. Check the online developer documentation for your device to find out if emulator definitions are available for download and installation into the AVD environment. When launched, an AVD will appear as a window containing an emulated Android device environment. for example, shows an AVD session configured to emulate the Google Nexus 7 device.

New AVDs are created and managed using the Android Virtual Device Manager, which may be used either in command-line mode or with a more user-friendly graphical user interface. Once the installation is complete, review the package list and make sure that the selected packages are now listed as Installed in the Status column. If any are listed as Not installed, make sure they are selected and click on the Install packages... button

CHAPTER 3

ANDROID APPLICATIONS

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file `AndroidManifest.xml` that describes each component of the application and how they interact. There are following four main components that can be used within an Android application. And the representation is shown below in the table

s.no	Components & Description
1	Activities They dictate the UI and handle the user interaction to the smart phone screen.
2	Services They handle background processing associated with an application.
3	Broadcast Receivers They handle communication between Android OS and applications.
4	Content Providers They handle data and database management issues.

TABLE NO 3.0 APPLICATIONS OF ANDROID APP

Activities

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

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

```
PublicclassMain ActivityextendsActivity  
  
{  
  
}
```

Services

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

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

```
PublicclassMyServiceextendsService  
  
{  
  
}
```

Broadcast Receivers

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

receiver is implemented as a subclass of **Broadcast Receiver** class and each message is broadcaster as an **Intent** object.

```
PublicclassMyReceiverextendsBroadcastReceiver
{
Publicvoid onReceive (context,intent) {}
}
```

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the Content resolver class. The data may be stored in the file system, the database or somewhere else entirely. A content provider is implemented as a subclass of content provider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
PublicclassMy ContentProviderextendsContentProvider
{
Publicvoid onCreate(){}
}
```

3.1. ADDITIONAL COMPONENTS

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are shown in the table 3.1

S.No	Components & Description
1	Fragments Represents a portion of user interface in an Activity.
2	Views UI elements that are drawn on-screen including buttons, lists forms etc.
3	Layouts View hierarchies that control screen format and appearance of the views.
4	Intents Messages wiring components together.
5	Resources External elements, such as strings, constants and drawable pictures.
6	Manifest Configuration file for the application.

TABLE NO 3.1 ADITIONAL COMPONENTS

RESOURCES ORGANIZING AND ACCESSING

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, colors,

layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under directory of the project. This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications. And shown below in the table 3.3.

TABLE NO 3.1.1 RESOURCES ORGANIZING AND ACCESSING

S.No.	Directory & Resource Type
1	anim/ XML files that define property animations. They are saved in res/anim/ folder and accessed from the R. anim class.
2	color/ XML files that define a state list of colors. They are saved in res/color/ and accessed from the R. color class.
3	drawable/ Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the R. drawable class.
4	layout/ XML files that define a user interface layout. They are saved in res/layout/ and accessed from the R. layout class.
5	menu/ XML files that define application menus, such as an Options Menu,

	Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the R. menu class.
6	raw/ Arbitrary files to save in their raw form. You need to call Resources.openraw Resources () with the resource ID, which is R.raw.Filename() to open such raw files.
7	values/ XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory – • arrays.xml for resource arrays, and accessed from the R. array class. • integers.xml for resource integers, and accessed from the R. integer class. • bools.xml for resource Boolean, and accessed from the R. bool class. • colors.xml for color values, and accessed from the R. color class. • dimens.xml for dimension values, and accessed from the R. dimen class. • strings.xml for string values, and accessed from the R. string class. • styles.xml for styles, and accessed from the R. style class.
8	xml/

Arbitrary XML files that can be read at runtime by calling <code>Resources.GetXML</code> . You can save various configuration files here which will be used at run time.
--

ALTERNATIVE RESOURCES

our application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e. Images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps –

- Here resources name will be any of the resources mentioned in the above table, like layout, drawable etc. The qualifier will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.
- Save the respective alternative resources in this new directory. The resource files must be named the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example, though image file name will be same but for high resolution screen, its resolution will be high.

3.2ACCESSING RESOURCES

During your application development, you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios our application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e. Images) for different screen resolution and alternative string resources for different languages.

At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps –

- Create a new directory in res/ named in the form <resources_name>-<config_qualifier>. Here resources_name will be any of the resources mentioned in the above table, like layout, drawable etc. The qualifier will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.
- Save the respective alternative resources in this new directory. The resource files must be named the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example, though image file name will be same but for high resolution screen, its resolution will be high.

3.3. OVER VIEW OF APPLICATIONS

An activity represents a single screen with a user interface just like window or frame of java. Android activity is the subclass of Context Theme Wrapper class. If you have worked with C, C++ or Java programming language then you must have seen that your program starts from main () function. Very similar way, Android system initiates its program with in an Activity starting with a call on create () callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram.

The activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implements those that ensure your app behaves the way user expect .And it can referred in the table.

TABLE NO 3.3OVERVIEW OF APPLICATIONS

S. No	Overview of Application
1	on Create () This is the first callback and called when the activity is first created.
2	on Start () This callback is called when the activity becomes visible to the user.
3	OnResume() This is called when the user starts interacting with the application.
4	OnPause() The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	OnStop() This callback is called when the activity is no longer visible.
6	OnDestroy() This callback is called before the activity is destroyed by the system.
7	OnRestart() This callback is called when the activity restarts after stopping it.

3.4. SERVICES

A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states –It can be show in the above table 3.3A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with start Service() and the diagram on the right shows the life cycle when the service is created with bind Service(). And show in the figure 3.1.

TABLE NO 3.4 SERVICES

S.No.	State & Description
1	Started A service is started when an application component, such as an activity, starts it by calling Startservice().Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	Bound A service is bound when an application component binds to it by calling BINDSERVICE A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

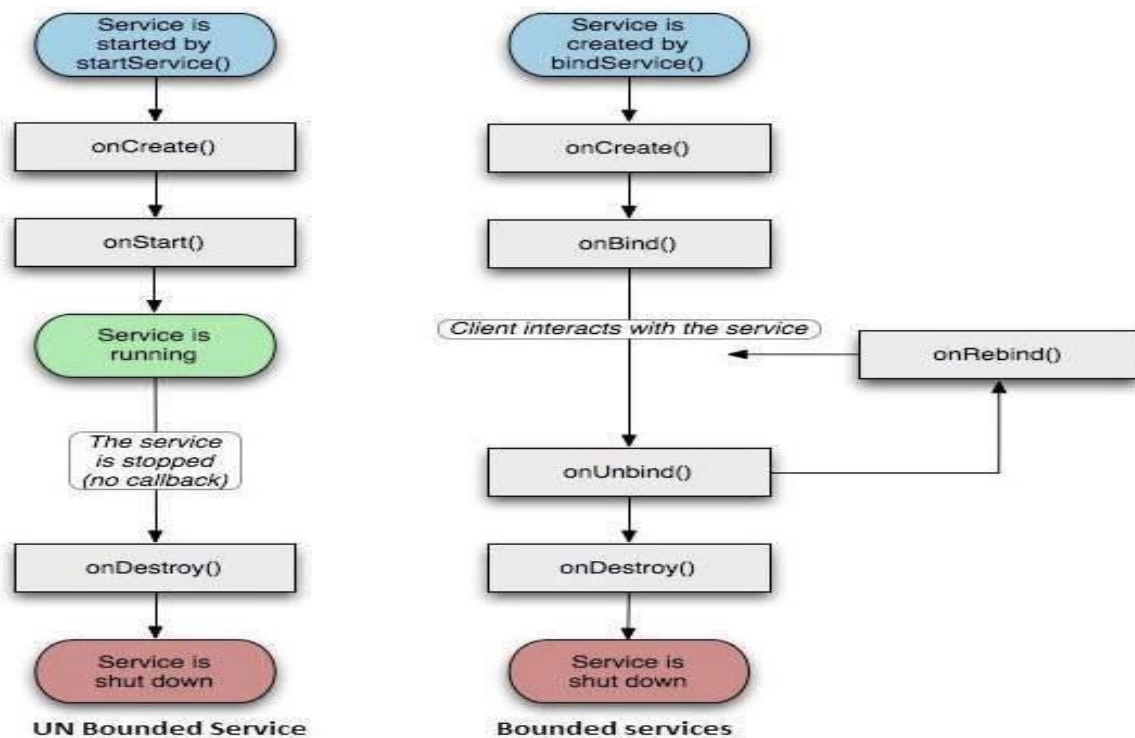


FIG 3.1 TYPES OF SERVICES

To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The Service base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect. It can be referred in the table 3.4.1

TABLE NO 3.4.1 CALLBACK AND DESCRIPTION

S. No.	Callback & Description
1	<code>OnStartCommand()</code> The system calls this method when another component, such as an activity, requests that the service be started, by calling <code>startService()</code>. If you implement this method, it is your responsibility to stop the service when its work is done, by calling <code>stopSelf()</code> or <code>stopService()</code> methods.

2	OnBind() The system calls this method when another component wants to bind with the service by calling <code>bindService()</code>. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an <code>IBinder</code> object. You must always implement this method, but if you don't want to allow binding, then you should return <code>null</code>.
3	OnUnbind() The system calls this method when all clients have disconnected from a interface published by the service.
4	OnRebind() The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its <code>onUnbind(Intent)</code>.
5	OnCreate() The system calls this method when the service is first created using <code>onStartCommand()</code> or <code>onBind()</code>. This call is required to perform one-time set-up.
6	OnDestroy() The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The Service base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect. Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadCastReceiver works for the system broadcasted intents –

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents. It can be shown in the figure 3.2

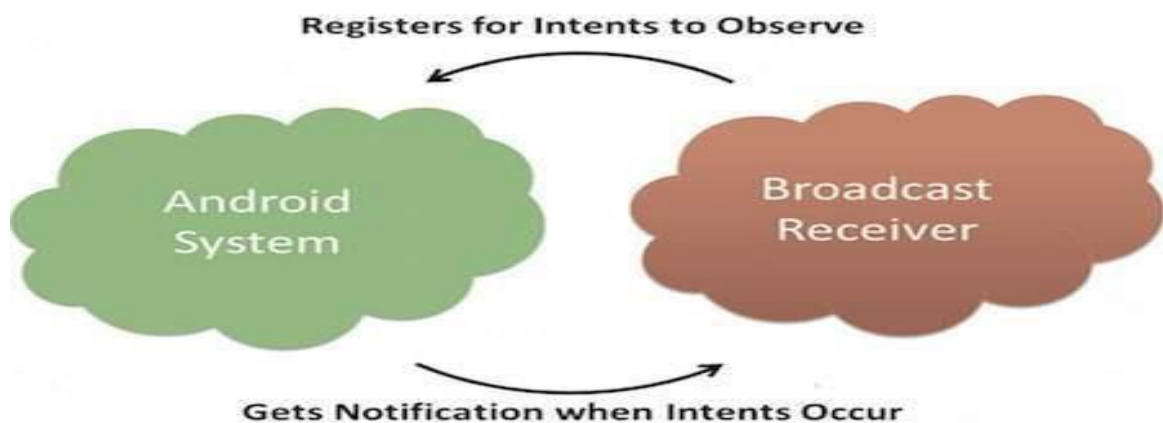


FIG 3.2 BROADCAST SERVICES

3.5. CONTENT PROVIDERS

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the `ContentResolver` class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network. It can be shown in the below figure 3.3. It is required to share data across applications. This is where content providers become very useful.

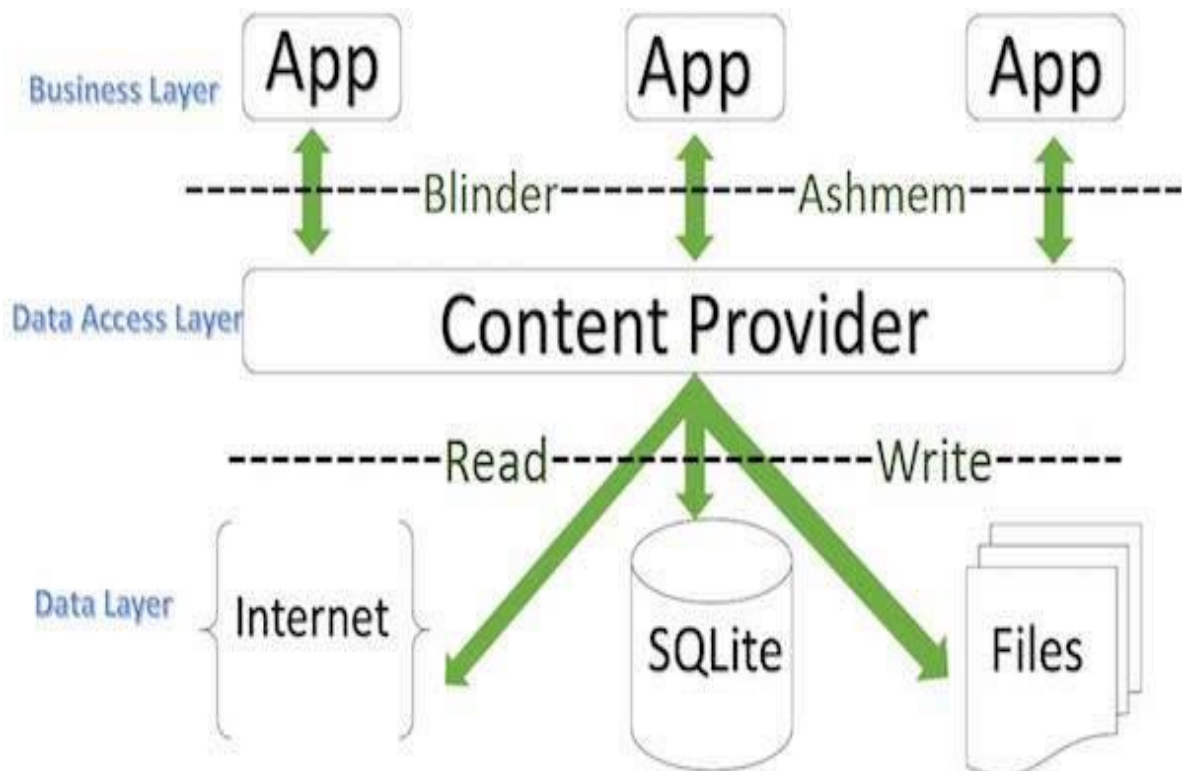


FIG 3.3 CONTENT PROVIDERS

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using `insert()`, `update()`, `delete()`, and `query()` methods. In most cases this data is stored in an SQLite database. A content provider is implemented as a subclass of `ContentProvider` class and must implement a standard set of APIs that enable other applications to perform transactions.

```
PublicclassMyApplicationextendsContentProvider
{
}Create Content Provider
```

This involves number of simple steps to create your own content provide the ContentProviderbase class.

- Second, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override onCreate() method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the onCreate() handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider>tag. Here is the list of methods which you need to override in Content Provider class to have your Content Provider working –in the figure below 3.4



FIG 3.4 CREATE CONTENT PROVIDERS

- onCreate() This method is called when the provider is started.
- Query() This method receives a request from a client. The result is returned as a Cursor object.

- Insert() This method inserts a new record into the content provider.
- Delete() This method deletes an existing record from the content provider.
- Update() This method updates an existing record from the content provider.
- getType() This method returns the MIME type of the data at the given URI.

3.6. FRAGMENTS

A Fragment is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of sub-activity.

Following are important points about fragment –

- A fragment has its own layout and its own behavior with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-plane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behavior that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element. Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So, we were not able to divide device screen and control different parts

separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design. In the below figure 3.5.

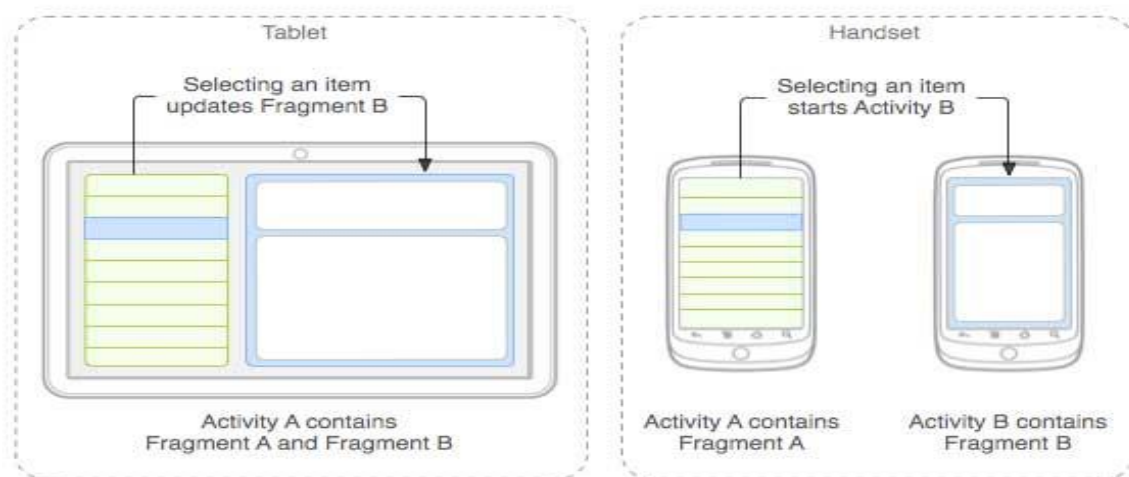


FIG 3.5. FRAGMENTS

The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle. it can be referred in the below figure 3.6.

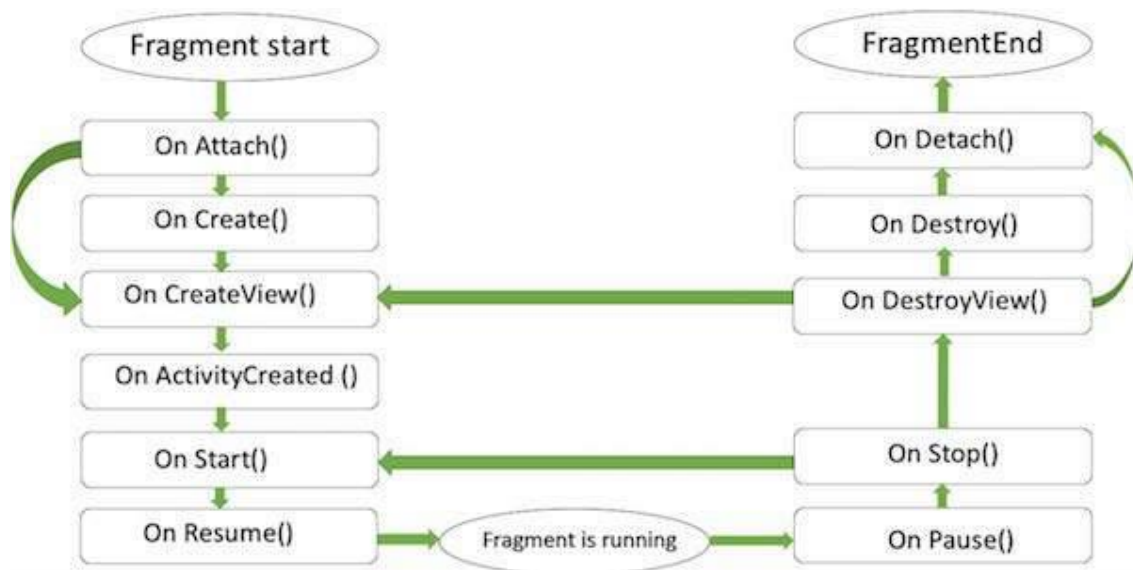


FIG .3.6. FRAGMENTLIFE CYCLE

Here is the list of methods which you can to override in your fragment class

- **OnAttach():** The fragment instance is associated with an activity instance. The fragment and the activity is not fully initialized. Typically, you get in this method a reference to the activity which uses the fragment for further initialization work.
- **OnCreate():** The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
- **OnCreateView():** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
- **OnActivityCreated():** The onActivityCreated () is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method, you can instantiate objects which require a Context object

- **OnStart():** The onStart() method is called once the fragment gets visible.
- **OnResume():** Fragment becomes active.
- **OnPause():** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
- **OnStop():** Fragment going to be stopped by calling onStop()
- **OnDestroyView():** Fragment view will destroy after calling this method
- **OnDestroy():** called to do final cleanup of the fragment's state but Not guaranteed to be called by the Android platform.
- For example—to continue with the news application example—the application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article. Thus, the application supports both tablets and handsets by reusing fragments in different combinations.

CHAPTER 4

SDK TOOLS

SDK tools are generally platform independent and are required no matter which android platform you are working on. When you install the Android SDK into your system, these tools get automatically installed. The list of SDK tools has been given below: The android developer tools lets you create interactive and powerful application for android platform. The tools can be generally categorized into two types. And it can be shown in below table 4.1.

- SDK tools
- Platform tools

TABLE NO 4.0. SDK TOOLS

S. NO	DESCRIPTION
1.	ANDROID This tool lets you manage AVDs, projects, and the installed components of the SDK.
2.	DDMS This tool lets you debug Android applications.
3.	DRAW 9 PATCHES This tool allows you to easily create a Nine Patch graphic using a WYSIWYG editor.

4.	EMULATOR This tool let you test your applications without using a physical device.
5.	MKSD CARD Helps you create a disk image (external SD storage) that you can use with the emulator.
6.	PROGUARD Shrinks, optimizes, and obfuscates your code by removing unused
7.	SQLTE Let's you access the SQLite data files created and used by Android applications.

4.1. PLATFORM TOOLS

The platform tools are customized to support the features of the latest androidplatform. The platform tools are typically updated every time you install a new SDKplatform. Each update of the platform tools is backward compatible with older

platforms. Some of the platform tools are listed below:

- Android Debug bridge (ADB)
- Android Interface definition language (AIDL)

- adapt, dexdump, and dex etc

4.2. INTERNAL STORAGE

Android provides many kinds of storage for applications to store their data. The storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection. We are going to look at the internal storage in this chapter. Internal storage is the storage of the private data on the device memory. By default, these files are private and are accessed by only your application and get deleted, when user delete your application.

Writing file

To use internal storage to write some data in the file, call the `openFileOutput()` method with the name of the file and the mode. The mode could be private, public etc.

Its syntax is given below:

```
FileOutputStream fout = openFileOutput("filenamehere", MODE_WORLD_READABLE);
```

The method `openFileOutput()` returns an instance of `FileOutputStream`. So, you receive it in the object of `FileInputStream`.

Reading file

To read from the file, you just created, call the `openFileInput()` method with the name of the file. It returns an instance of `FileInputStream`. Its syntax is given below:

```
FileInputStream fin = openFileInput(file);
```

After that, you can call `read` method to read one character at a time from the file and then you can print.

CHAPTER 5

APPENDIX

5.1. SCREENSHOTS

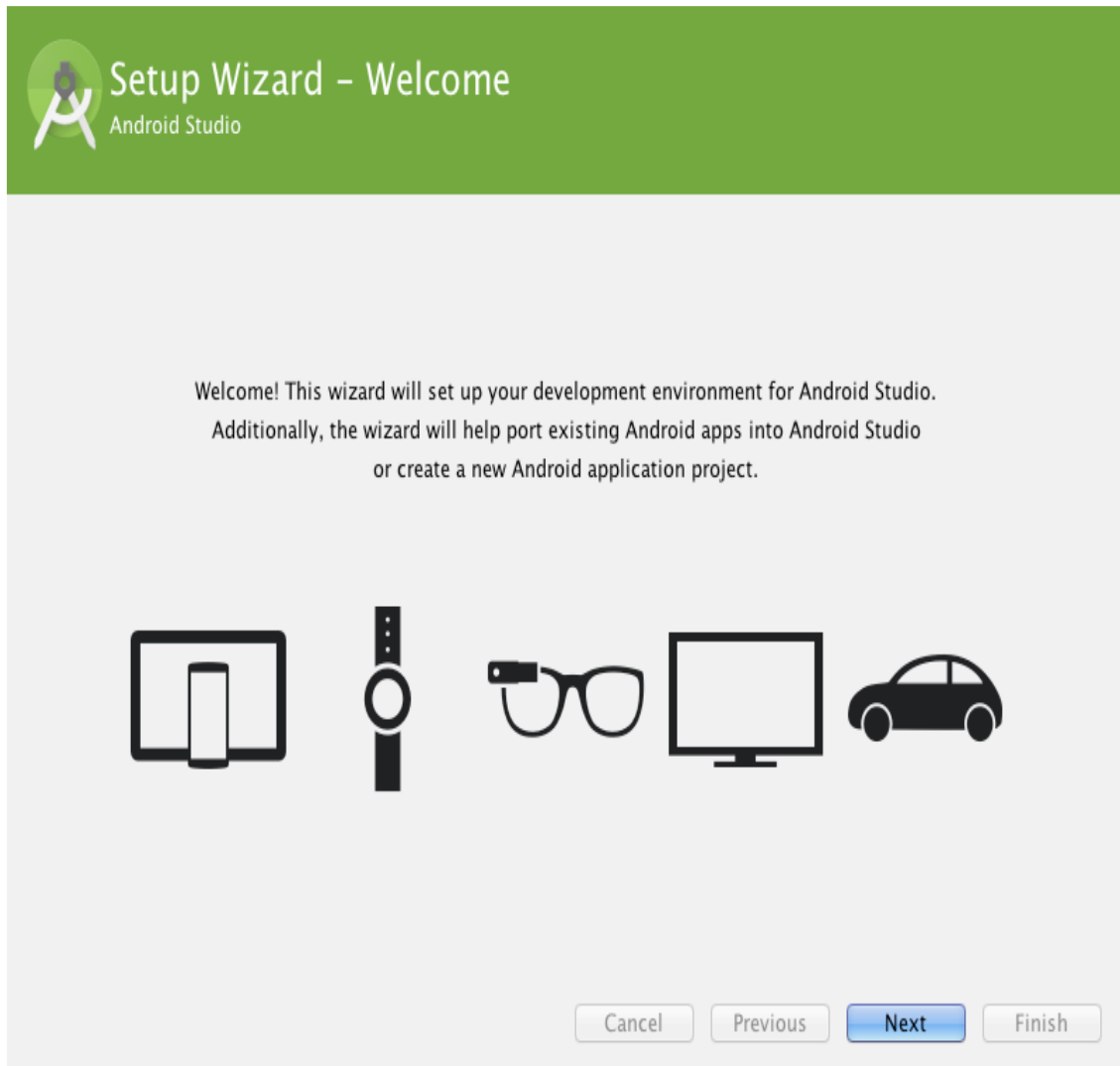
First we need to install JDK to support android studio kit.



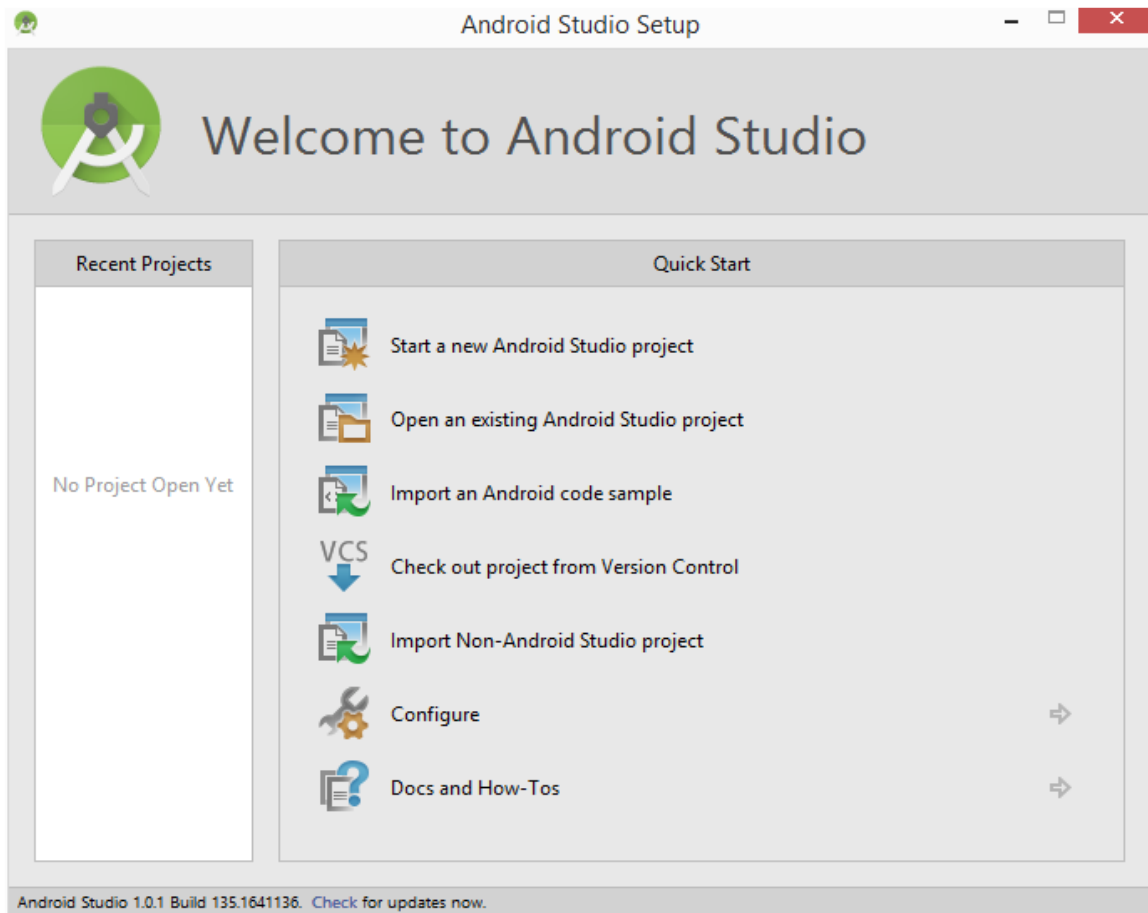
This is android studio software where android app can be developed.



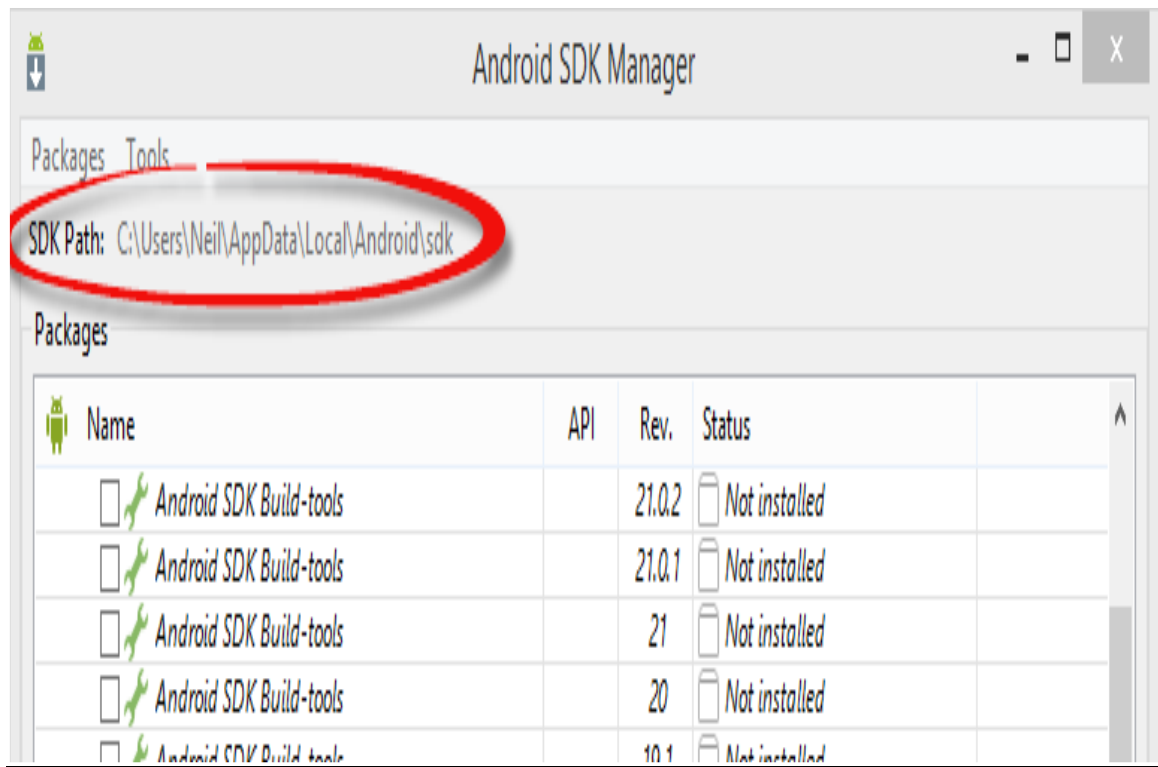
Setting up a wizard.



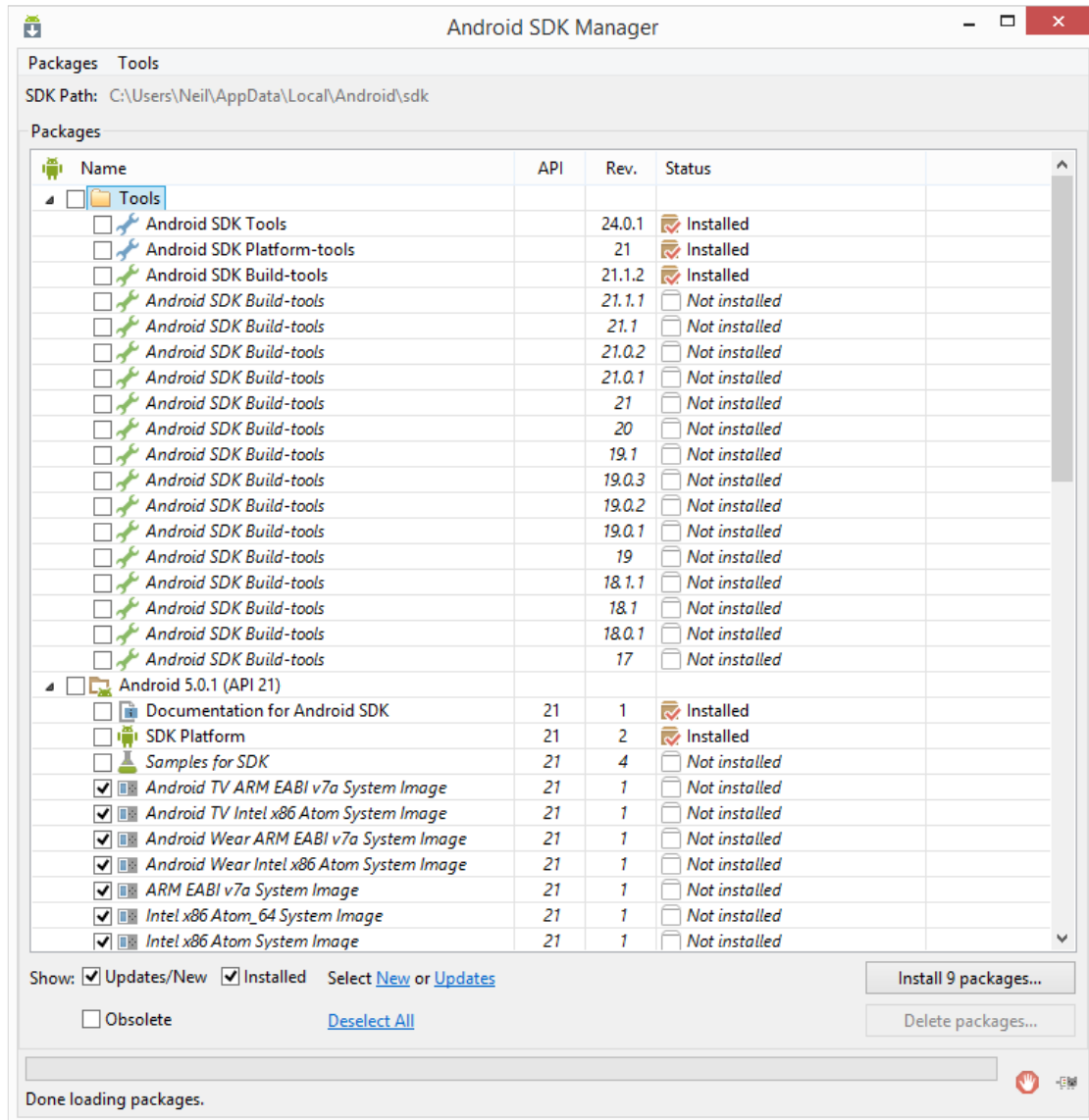
SETTING UP AN ANDROID STUDIO APP ENVIRONMENT



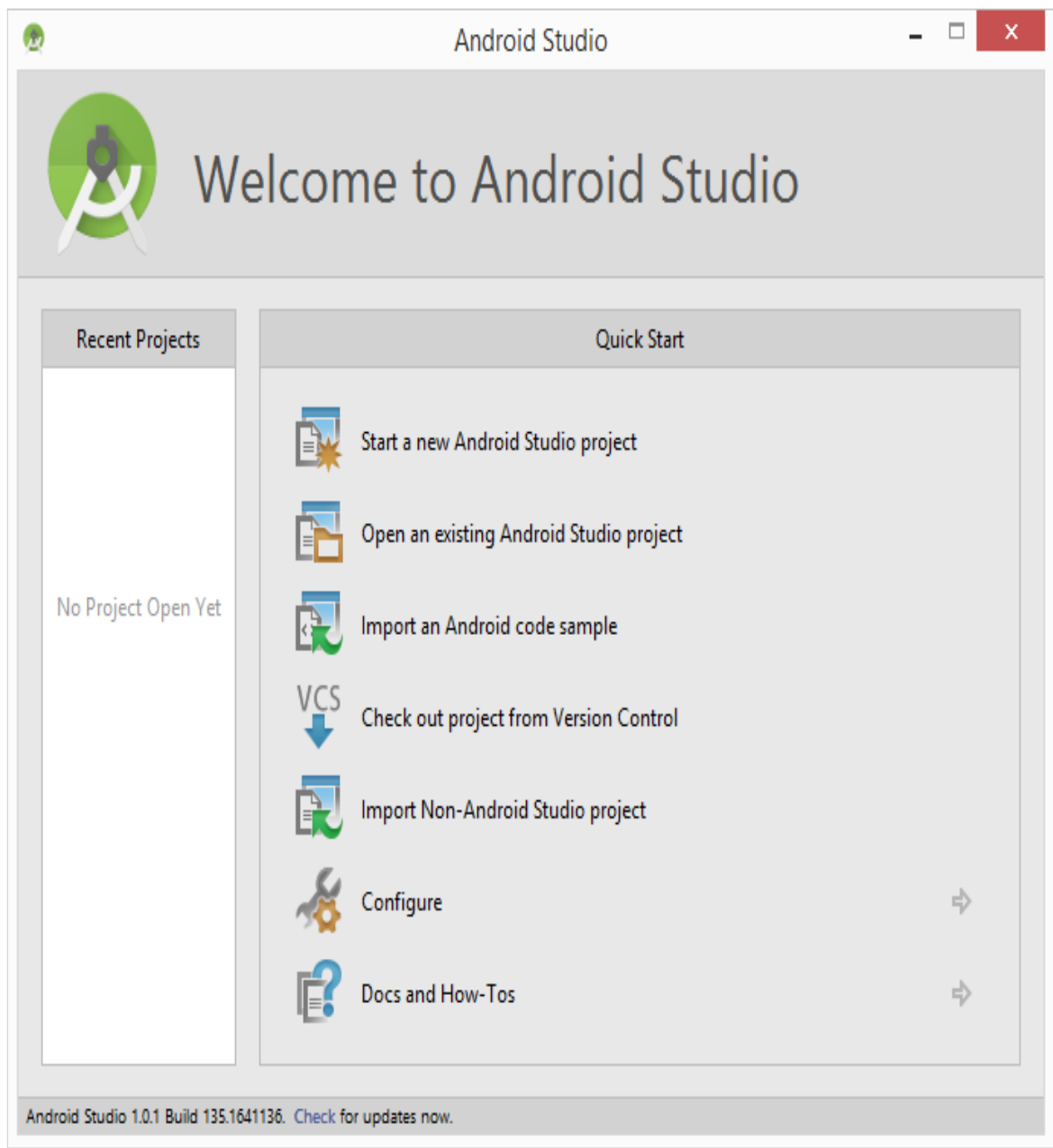
MAKING COMMAND LINE ACCESSIBLE



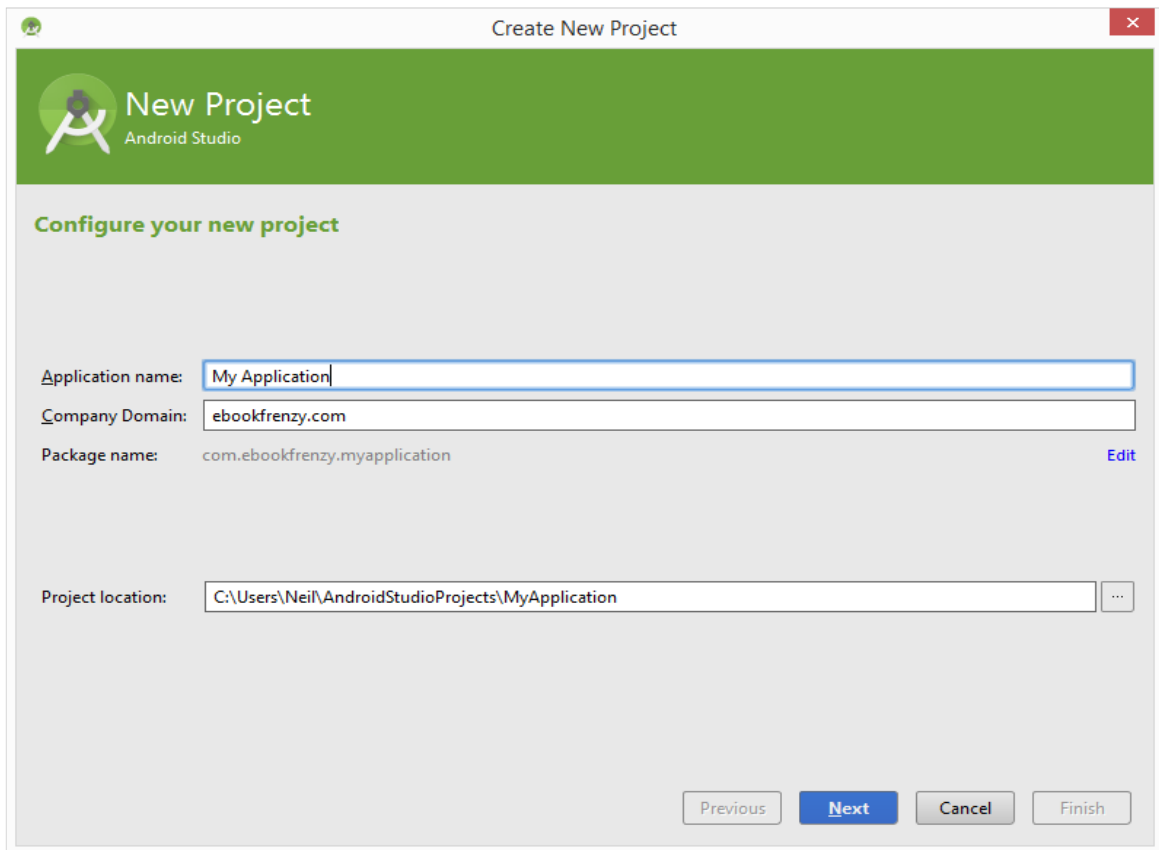
INSTALLING THE ANDROID SDK PACKAGES



CREATING A NEW ANDROID PROJECT



PROCESS FOR DOING A NEW PROJECT



Create New Project

New Project
Android Studio

Configure your new project

Application name: My Application

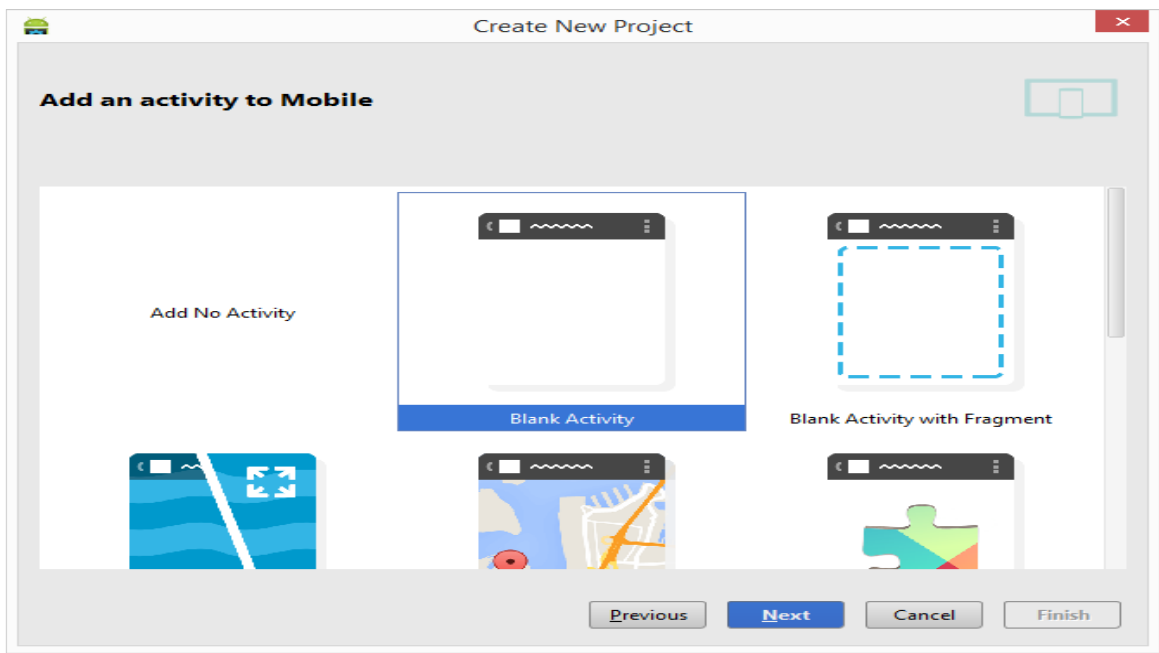
Company Domain: ebookfrenzy.com

Package name: com.ebookfrenzy.myapplication [Edit](#)

Project location: C:\Users\Neil\AndroidStudioProjects\MyApplication

Previous Next Cancel Finish

CREATING AN NEW ACTIVITY



Create New Project

Add an activity to Mobile

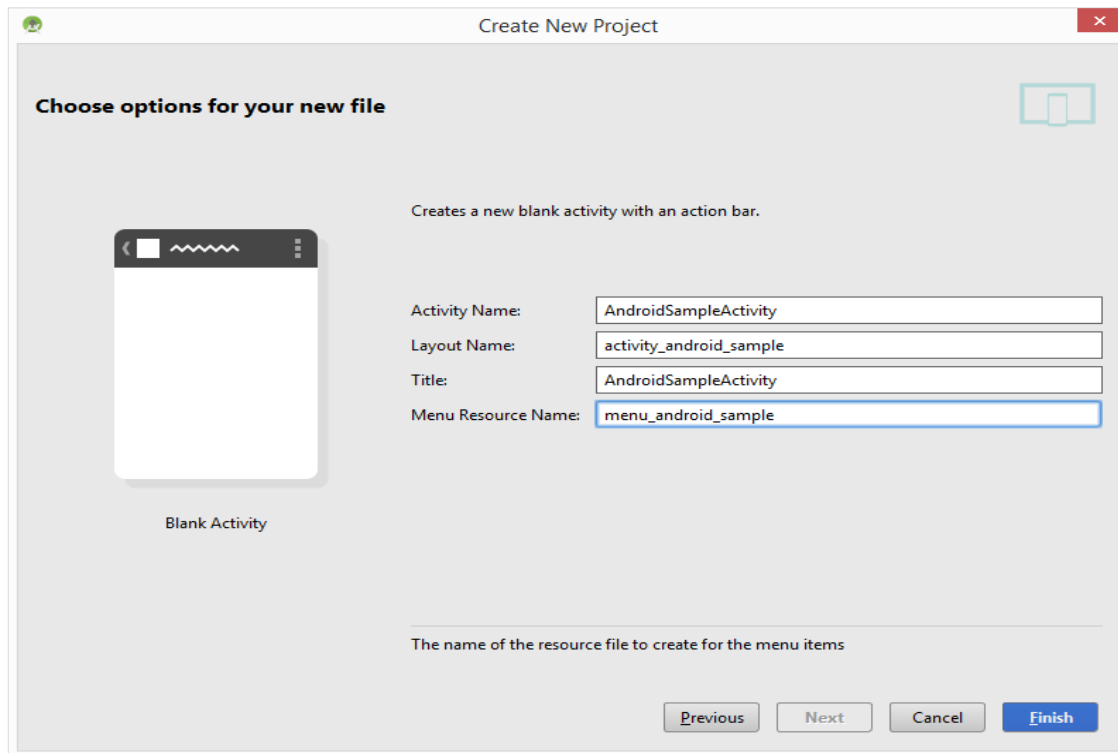
Add No Activity

Blank Activity

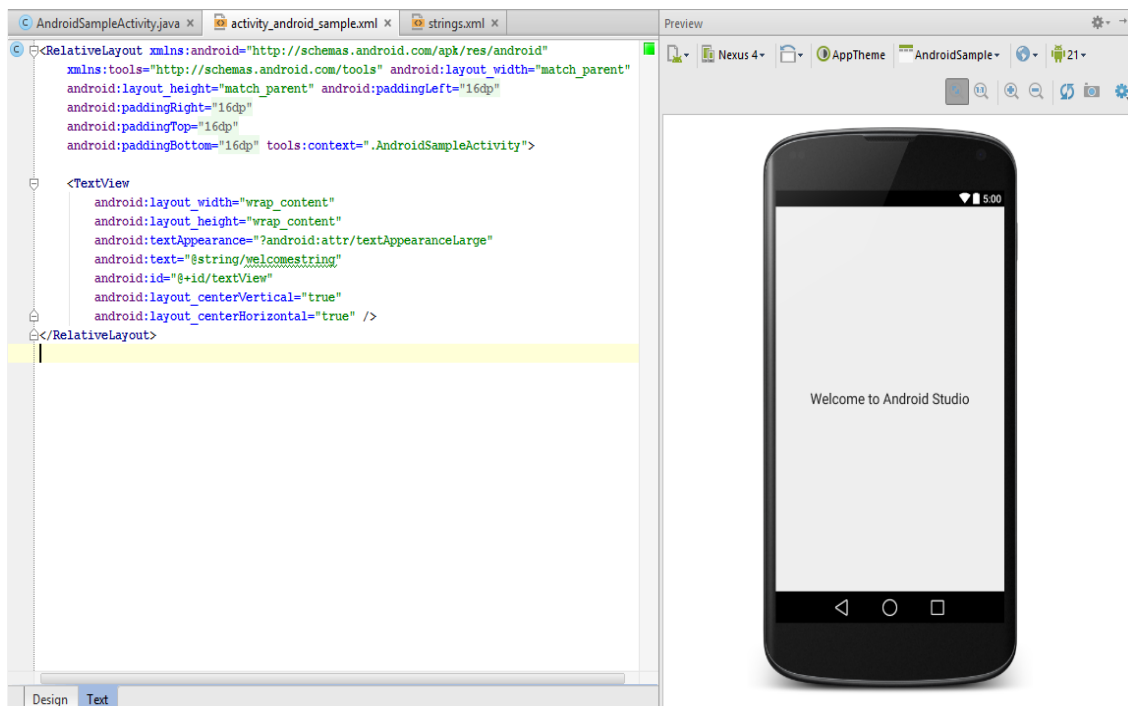
Blank Activity with Fragment

Previous Next Cancel Finish

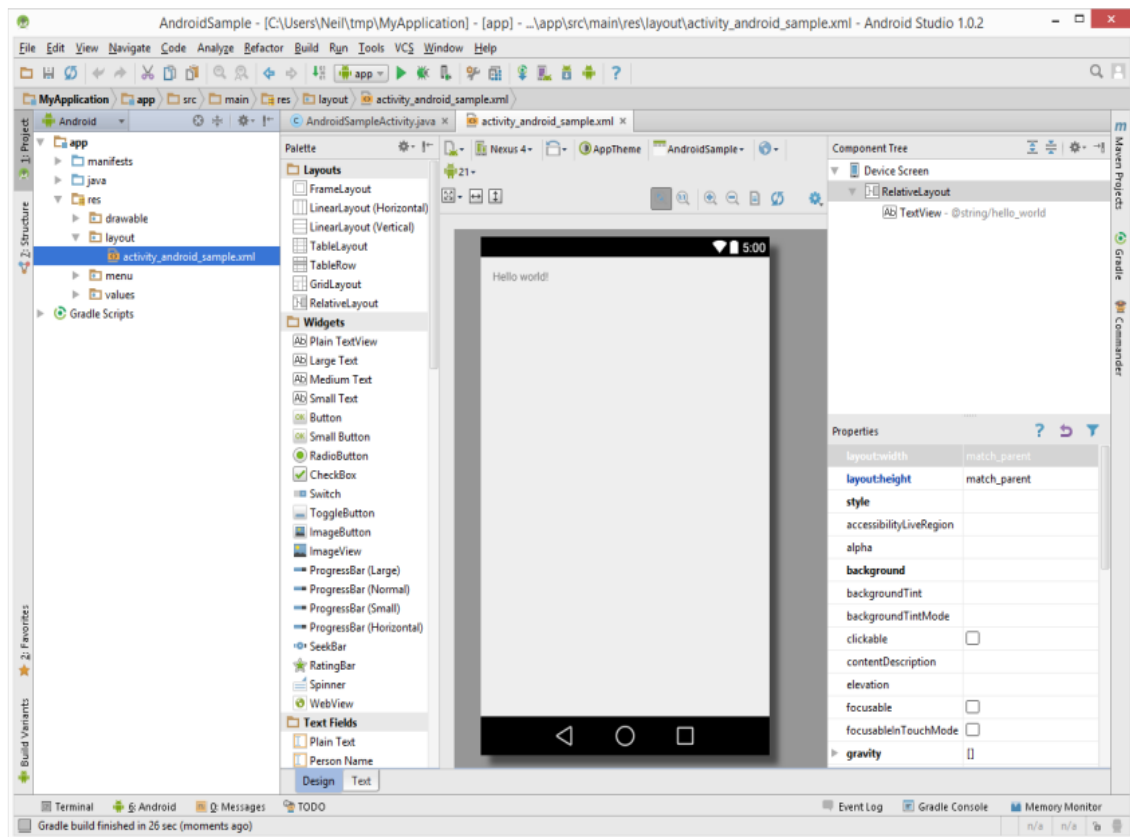
CREATING AN ANDROID APP IN ANDROID SDK



SWITCHING YOUR SOURCE CODE TO THE PHONE



MODELLING AN EXMAPLE HAVING TOOLS



6. SOURCE CODE FOR CALCULATOR

```
package com.sample.foo.samplecalculator;

import android.databinding.DataBindingUtil;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

import com.sample.foo.samplecalculator.databinding.ActivityMainBinding;

import java.text.DecimalFormat;

public class MainActivity extends AppCompatActivity {

    private ActivityMainBinding binding;

    private static final char ADDITION = '+';
    private static final char SUBTRACTION = '-';
    private static final char MULTIPLICATION = '*';
    private static final char DIVISION = '/';

    private char CURRENT_ACTION;

    private double valueOne = Double.NaN;
    private double valueTwo;

    private DecimalFormat decimalFormat;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        decimalFormat = new DecimalFormat("#.#####");

        binding = DataBindingUtil.setContentView(this, R.layout.activity_main);

        binding.buttonDot.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                binding.editText.setText(binding.editText.getText() + ".");
            }
        });

        binding.buttonZero.setOnClickListener(new View.OnClickListener() {
            @Override
```

```

        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "0");
        }
    });

    binding.buttonOne.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "1");
        }
    });

    binding.buttonTwo.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "2");
        }
    });

    binding.buttonThree.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "3");
        }
    });

    binding.buttonFour.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "4");
        }
    });

    binding.buttonFive.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "5");
        }
    });

    binding.buttonSix.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            binding.editText.setText(binding.editText.getText() + "6");
        }
    });

    binding.buttonSeven.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

```

```

        binding.editText.setText(binding.editText.getText() + "7");
    }
});

binding.buttonEight.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        binding.editText.setText(binding.editText.getText() + "8");
    }
});

binding.buttonNine.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        binding.editText.setText(binding.editText.getText() + "9");
    }
});

binding.buttonAdd.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        computeCalculation();
        CURRENT_ACTION = ADDITION;
        binding.infoTextView.setText(decimalFormat.format(valueOne) + "+");
        binding.editText.setText(null);
    }
});

binding.buttonSubtract.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        computeCalculation();
        CURRENT_ACTION = SUBTRACTION;
        binding.infoTextView.setText(decimalFormat.format(valueOne) + "-");
        binding.editText.setText(null);
    }
});

binding.buttonMultiply.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        computeCalculation();
        CURRENT_ACTION = MULTIPLICATION;
        binding.infoTextView.setText(decimalFormat.format(valueOne) + "*");
        binding.editText.setText(null);
    }
});

binding.buttonDivide.setOnClickListener(new View.OnClickListener() {
    @Override

```

```

        public void onClick(View view) {
            computeCalculation();
            CURRENT_ACTION = DIVISION;
            binding.infoTextView.setText(decimalFormat.format(valueOne) + "/"");
            binding.editText.setText(null);
        }
    });

    binding.buttonEqual.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            computeCalculation();
            binding.infoTextView.setText(binding.infoTextView.getText().toString() +
                decimalFormat.format(valueTwo) + " = " +
decimalFormat.format(valueOne));
            valueOne = Double.NaN;
            CURRENT_ACTION = '0';
        }
    });

    binding.buttonClear.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if(binding.editText.getText().length() > 0) {
                CharSequence currentText = binding.editText.getText();
                binding.editText.setText(currentText.subSequence(0,
currentText.length()-1));
            }
            else {
                valueOne = Double.NaN;
                valueTwo = Double.NaN;
                binding.editText.setText("");
                binding.infoTextView.setText("");
            }
        }
    });
}

private void computeCalculation() {
    if(!Double.isNaN(valueOne)) {
        valueTwo = Double.parseDouble(binding.editText.getText().toString());
        binding.editText.setText(null);

        if(CURRENT_ACTION == ADDITION)
            valueOne = this.valueOne + valueTwo;
        else if(CURRENT_ACTION == SUBTRACTION)
            valueOne = this.valueOne - valueTwo;
        else if(CURRENT_ACTION == MULTIPLICATION)
            valueOne = this.valueOne * valueTwo;
        else if(CURRENT_ACTION == DIVISION)

```

```
        valueOne = this.valueOne / valueTwo;
    }
    else {
        try {
            valueOne = Double.parseDouble(binding.editText.getText().toString());
        }
        catch (Exception e){}
    }
}
}
```

7.REFERENCES

- 1.<https://developer.android.com/reference/packages.html>
 - 2.<http://developer.android.com/reference/java/lang/ref/refernce.html>
 - 3.http://www.sersc.org/journals/IJMUE/vol9_no4_2014/20.pdf
 - 4.<https://www.gov.uk/government/publications/androidapplication-development-guidance/android-application-development-guidance-in-GOV.UK>.
 - 5.<https://www.inf.ed.ac.uk/publications/thesis/online/IM100767.pdf>.
 - 6.<http://ijcttjournal.org/Volume3/issue-3/IJCTT-V3I3P130.pdf>
 7. Garimapandey, Dikshadani , "Department-android application using an eclipse Department of Computer Science & Eng, Mahamay Technical University, Noida , Department International Journal of Scientific and Research Publications, Vol. 4, 2014,pp.1-5.
- [1] Albert Mayan.J,Surya.B,Pranoy Prabhakar,Princekumar,"Department–Student Library Using Twig Pattern Query Processing Over Admin-UserLogin Privilege", Pakistan Journal of Biotechnology., Vol. 13 , 2016