INTRODUCTION TO ANDROID

INTRODUCTION:

Android is a system developed by Google, based on a modified version of the kernel and other source software and designed primarily for touch screen mobile devices such as smart phones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. Variants of Android are also used on consoles, digital, PCs and other electronics.

The Android Operating System is a Linux-based OS developed by the Open Handset Alliance (OHA). The Android OS was originally created by Android, Inc., which was bought by Google in 2005. Google teamed up with other companies to form the Open Handset Alliance (OHA), which has become responsible for the continued development of the Android OS.

The android is a powerful operating system and it supports large number of applications in Smart phones. These applications are more comfortable and advanced for the users. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it's free and any one can use it.

The android has got millions of apps available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular.

Each time the OHA releases an Android version, it names the release after a dessert. Android 1.5 is known as Cupcake, 1.6 as Donut, 2.0/2.1 as Eclair, 2.2 as Froyo and 2.3 is dubbed Gingerbread. Once a version is released, so is its source code.

Android versions:

Google did not attach any high-calorie code name to its initial versions 1.0 and 1.1 of the Android Operating System. The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat, Lollipop and Marshmallow. Let's understand the android history in a sequence.

THE ANDROID 4.1 JELLY BEAN SDK

The Android 4.1 Jelly Bean SDK was released with new features for developers in July 2012. It improves the beauty and simplicity of Android 4.0 and is a major platform release that adds a variety of new features for users and app developers. A few of the big features of this release include the following:

- **Project Butter**: Makes the Jelly Bean UI faster and more responsive. Also CPU Touch Responsiveness is added, which increases CPU performance whenever the screen is touched. It uses the finger's speed and direction to predict where it will be located after some milliseconds, hence making the navigation faster.
- **Faster speech recognition**: Speech recognition is now faster and doesn't require any network to convert voice into text. That is, users can dictate to the device without an Internet connection.
- **Improved notification system:** The notifications include pictures and lists along with text. Notifications can be expanded or collapsed through a variety of gestures, and users can block notifications if

desired. The notifications also include action buttons that enable users to call directly from the notification menu rather replying to email.

- **Supports new languages:** Jelly Bean includes support for several languages including Arabic, Hebrew, Hindi, and Thai. It also supports bidirectional text.
- **Predictive keyboard:** On the basis of the current context, the next word of the message is automatically predicted.
- **Auto-arranging Home screen:** Icons and widgets automatically resize and realign as per the existing space.
- **Helpful for visually impaired users:** The Gesture Mode combined with voice helps visually impaired users to easily navigate the user interface.
- **Improved Camera app:** The Jelly Bean Camera app includes a new review mode of the captured photos. Users can swipe in from the right of the screen to quickly view the captured photos. Also, users can pinch to switch to a new film strip view, where they can swipe to delete photos.
- **Better communication in Jelly Bean:** Two devices can communicate with Near Field Communication (NFC); that is, two NFC-enabled Android devices can be tapped to share data. Also, Android devices can be paired to Bluetooth devices that support the Simple Secure Pairing standard by just tapping them together.
- **Improved Google Voice search:** Jelly Bean is equipped with a question and answer search method that helps in solving users' queries similar to Apple's popular Siri.
- **Face Unlock:** Unlocks the device when the user looks at it. It also prevents the screen from blacking out. Optionally "blink" can be used to confirm that a live person is unlocking the device instead of a photo.
- **Google Now:** Provides users "just the right information at just the right time." It displays cards to show desired information automatically. For example, the Places card displays nearby restaurants and shops while moving; the Transit card displays information on the next train or bus

when the user is near a bus stop or railway station; the Sports card displays live scores or upcoming game events; the Weather card displays the weather conditions at a user's current location, and so on.

- **Google Play Widgets:** Provides quick and easy access to movies, games, magazines, and other media on the device. It also suggests new purchases on Google Play.
- **Faster Google Search:** Google Search can be opened quickly, from the lock screen and from the system bar by swiping up and also by tapping a hardware search key if it is available on the device.
- **Supports antipiracy:** This feature supports developers in the sense that the applications are encrypted with a device-specific key making it difficult to copy and upload them to the Internet.

HISTORY OF MOBILE GENERATIONS :

History was created in April 1972 with first call made on a mobile phone. "G" refers to generation, related to next generation wireless technologies. Mobile phones started out coming in the market with Motorola introducing the first mobile phone on 3 April 1973. Generation of these phones were known as 0G mobile phones in which different technologies were used like push to talk, mobile telephone system, improved mobile telephone system etc. Martin Cooper, an employee of Motorola Company is considered as key player as he developed the first mobile phone. Before him, handsets were used in vehicles, but he came with the development of first truly portable mobile phone. Each of the generation is making us faster, secure as well as more reliable as compared to previous ones. It is hard to overcome this reliability factor. We see smart phones as our companions today and we are dependent on these devices in large number of ways.

A new subscriber signs up after every 2.5 seconds .With the increasing demands in the field of mobile and data communications, the sole aim is to connect users as fast as possible. In past few years, mobile wireless communication has experienced different generations of technology mainly from 0G to 5G. Future technologies such

as 6G and 7G has also been shown providing immense scope for innovative research and development.

COMPARISON OF ALL GENERATIONS OF MOBILE TECHNOLOGIES

Technology → / Features ↓	1G	2G	3G	4G	5G
Start/ Deployment	1970 – 1980	1990 - 2004	2004-2010	Now	Soon (probably 2020)
Data Bandwidth	2kbps	64kbps	2Mbps	1 Gbps	Higher than 1Gbps
Technology	Analog Cellular Technology	Digital Cellular Technology	CDMA 2000 (1xRTT, EVDO) UMTS, EDGE	WiMax LTE Wi-Fi	WWWW(coming soon)
Service	Mobile Telephony (Voice)	Digital voice, SMS, Higher capacity packetized data	Integrated high quality audio, video and data	Dynamic Information access, Wearable devices	Dynamic Information access, Wearable devices with AI Capabilities
Multiplexing	FDMA	TDMA, CDMA	CDMA	CDMA	CDMA
Switching	Circuit	Circuit, Packet	Packet	All Packet	All Packet
Core Network	PSTN	PSTN	Packet N/W	Internet	Internet

Wireless telephone started with what you might call 0G if you remember back that far. In those pre-cell days, you had a mobile operator to set up the calls and there were only a handful of channels available.

0G refers to pre-cell phone mobile telephony technology, such as radio telephones that some had in cars before the advent of cell phones. Mobile radio telephone systems preceded were the predecessors of the first generation of cellular telephones; these systems are called 0G (zero generation) systems. These early mobile telephone systems can be distinguished from earlier closed radiotelephone systems in that they were available as a commercial service that was part of the public switched telephone network, with their own telephone numbers, rather than part of a closed network such as a police radio or taxi dispatch system. These mobile telephones were usually mounted in cars or trucks, though briefcase models were also made.

FIRST GENERATION (1G):

The first generation of cellular systems used analog radio technology. Analog cellular systems consist of three basic elements: a mobile telephone (mobile radio), cell sites, and a mobile switching center (MSC). A mobile telephone communicates by radio signals to the cell site within a radio coverage area. The cell site s base station (BS) converts these radio signals for transfer to the MSC via wired (landline) or wireless (microwave) communications links. The MSC routes the call to another mobile telephone in the system or the appropriate landline facility. These three elements are integrated to form a ubiquitous coverage radio system that can connect to the public switched telephone network (PSTN). It support speed up to 2.4kbps.

SECOND GENERATION (2G):

It is based on GSM or in other words global system for mobile communication. It was launched in Finland in the year 1991. It was the first digital cellular networks, which had a number of obvious benefits over the analog networks they were supplanting: improved sound quality, better security, etc. 2G technologies have replaced the analogy technology by digital communication by providing services such as text message, picture message and MMS. All text messages are digitally encrypted in 2G technology. This digital encryption allows for the transfer of data in such a way that only the intended receiver can receive and read it. There are 3 different types (FDMA, TDMA/GSM, and CDMA) of 2G mobile technologies are designed with different working methods, properties and specifications.

THIRD GENERATION (3G):

The third generation of mobile systems provides high speed data transmissions of 144kbps and higher. It comes with enhancements over previous wireless technologies, like high-speed transmission, advanced multimedia access and global roaming. 3G is mostly used with mobile phones and handsets as a means to connect the phone to the Internet or other IP networks in order to make voice and video calls, to download and upload data and to surf the net. 3G will support multimedia applications such as full-motion video, video conferencing and Internet access. The

data are sent through the technology called Packet Switching Voice calls are interpreted through Circuit Switching. It is a highly sophisticated form of communication that has come up in the last decade.

FOURTH GENERATION (4G):

The Fourth Generation of mobile communication upgrade existing communication networks and is expected to provide a comprehensive and secure IP based solution where facilities such as voice, data and streamed multimedia will be provided to users on an "Anytime, Anywhere" basis and at much higher data rates compared to previous generations.

FIFTH GENERATION (5G):

In 5G, researches are related to the development of World Wide Wireless Web (WWWW), Dynamic adhoc Wireless Networks (DAWN) and Real Wireless Communication. The most important technologies for 5G technologies are 802.11 Wireless Local Area Networks (WLAN) and 802.16 Wireless Metropolitan Area Networks (WMAN), Ad-hoc Wireless Personal Area Network (WPAN) and Wireless networks for digital Communication. Some features of 5G Technology are given below:

5G is a completed wireless communication with almost no limitation; somehow people called it REAL wireless world Additional features such as Multimedia Newspapers, also to watch T.V programs with the clarity as to that of an HD T.V. We can send Data much faster than that of the previous generations. 5G will bring almost perfect real world wireless or called “WWWW: World Wide Wireless Web . Wearable devices with AI capabilities.

FEATURES OF ANDROID:

Access to hardware:

Android include API libraries which simplifies the access of device hardware. They ensure that you don’t need to create specific implementations of your software for

different devices that means no matter what ever device you are given with, if you use these API libraries, these API's will help you to create android application that will run on devices which support android stack

Maps and Location based Services:

OpenGL – Open Graphics Library :With the help of open GL we can design our own maps which are rich in graphics library with the help of these maps, we can interact with Google maps.

Geo Coding: Android uses location based services such as GPS and Google network based, location based technology to determine the devices current position. These services enforce an abstraction from specific location based technology and let you to specify minimum requirement

Eg: Accuracy of cost

SQLite Database-[Super Quotient Lite]

- It is an inbuilt database of android.
- It is data storage device.
- Rapid and efficient storage for those devices whose storage capacity is very limited
- This database is relational database and a sandbox.
- SQLite contains content provider which abstracts the features of the database from the user and at the same time providing security to the database.

Inter application communication and shared data:

- One application communicates with another application with the help of message passing mechanism called as intents
- Different application share common data. This sharing of data is done by services
- This services enables background processing
- Content provider is used to provide secure, managed access to your application private databases

Cloud to Device Messaging[C2DM]:

- With the help of C2DM you can always create an connection between your mobile application and your server.
- Whenever server gets new data C2DM services prompts the application in your mobile by sending this new data to the mobile.

Optimized memory & process management:

Android runtime manages the process lifetime i.e, Android stops or kills the processes as necessary and the forced resources are allocated for higher priority application and at the same time if necessary it will also restart those applications which are filled and update their resources.

Using widgets and live wallpaper to enhance the Home screen:

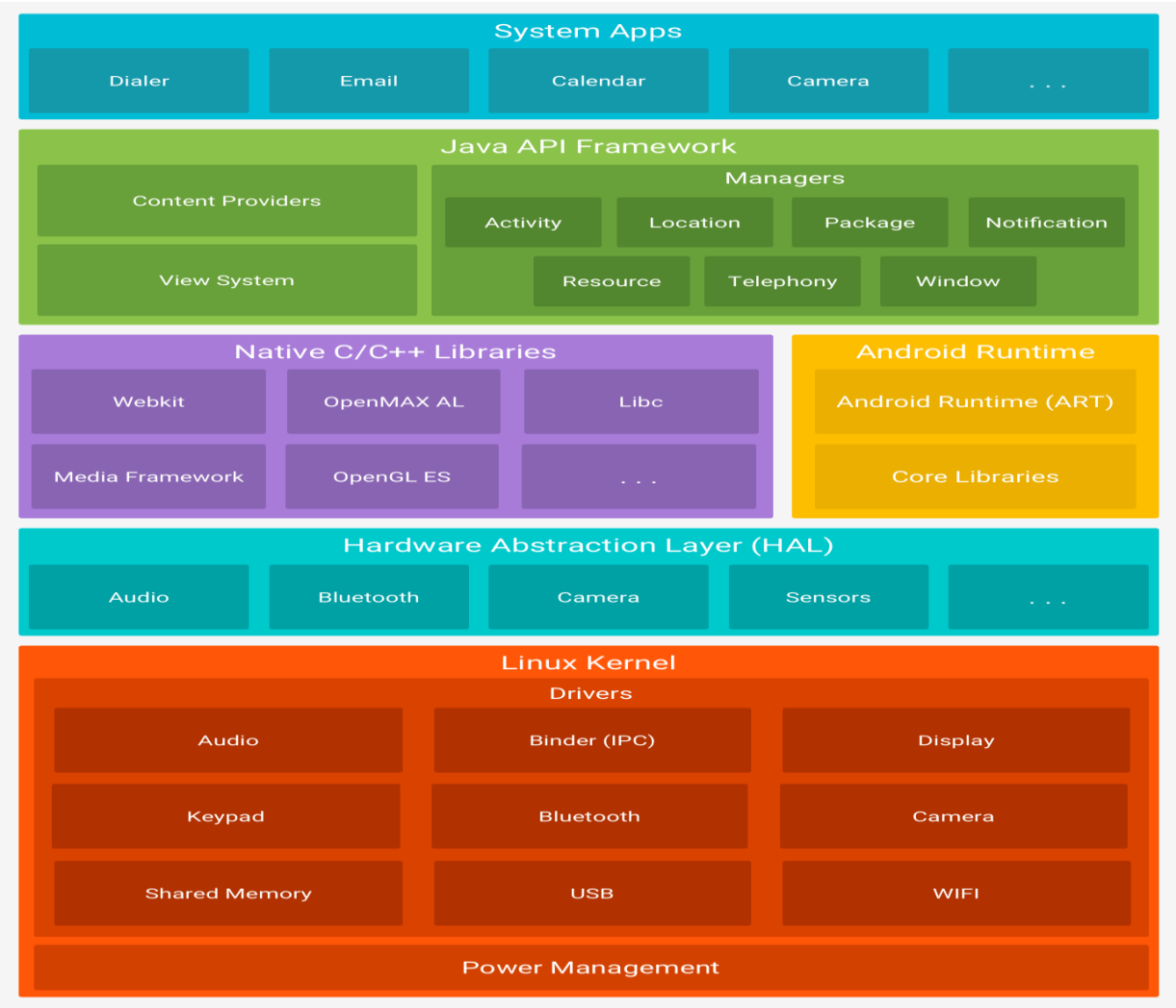
Widgets and live wallpapers let you create dynamic application components that provide a window into your application, or offer useful and timely information directly on the home screen.

Background services:

- Android supports applications and services designed to run in the background while your application is not being actively used
- Generally only one interactive application is visible at anytime
- Background services make it possible to create invisible application components that perform automatic processing without direct user action
- Notification alerts users to events that have happened in a background applications.

ANDROID ARCHITECTURE:

Android is an open source, Linux-based software stack created for a wide array of devices and form factors.



The Android software stack.

1. Linux kernel

It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. This layer is the foundation of the Android Platform contains all low level drivers for various hardware components support. Android Runtime relies on Linux Kernel for core system services like, Memory, process management, threading ,Network stack ,Driver model, Security and more.

2. Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data,

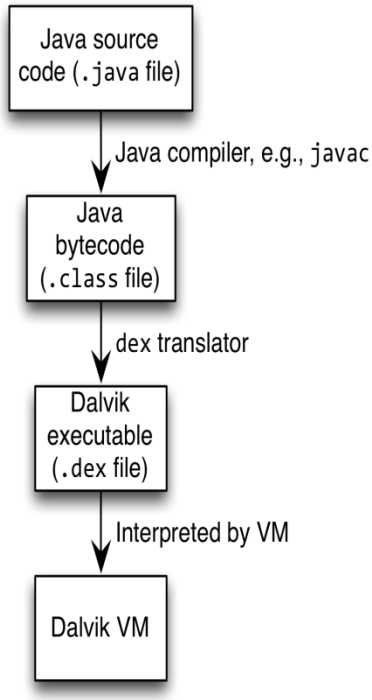
libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

- SQLite Library used for data storage and light in terms of mobile memory footprints and task execution.
- WebKit Library mainly provides Web Browsing engine and a lot more related features.
- The surface manager library is responsible for rendering windows and drawing surfaces of various apps on the screen.
- The media framework library provides media codecs for audio and video.
- The OpenGl (Open Graphics Library) and SGL(Scalable Graphics Library) are the graphics libraries for 3D and 2D rendering, respectively.
- The FreeType Library is used for rendering fonts.

3. Android Runtime:

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

- Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
- Android apps execute on Dalvik VM, a “clean-room” implementation of JVM
- Dalvik optimized for efficient execution
- Dalvik: register-based VM, unlike Oracle’s stack-based
- JVM
- Java .class bytecode translated to Dalvik EXecutable (DEX) bytecode, which Dalvik interprets



4. Android Framework

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

Activity Manager: manages the life cycle of an applications and maintains the back stack as well so that the applications running on different processes has smooth navigations.

Package Manager: keeps track of which applications are installed in your device.

Window Manager : Manages windows which are java programming abstractions on top of lower level surfaces provided by surface manager.

Telephony Managers: manages the API which is use to build the phone applications

Content Providers: Provide feature where one application can share the data with another application. like phone number , address, etc

View Manager : Buttons , Edit text , all the building blocks of UI, event dispatching etc.

5. Applications

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernel. Any applications that you write are located at this layer.

App Components Of Android

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four types of app components:

- activities
- Services
- Broadcast receivers
- Content providers

Each type serves a distinct purpose and has a distinct lifecycle that defines how a component is created and destroyed. The following sections describe the four types of app components.

Activities

An *activity* is the entry point for interacting with the user. It represents a single screen with a user interface.

For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others.

A different app can start any one of these activities if the email app allows it.

For example, a camera app might start the activity in the email app for composing a new email to let the user share a picture.

An activity facilitates the following key interactions between system and app:

- Keeping track of what the user currently cares about—what is on-screen—so that the system keeps running the process that is hosting the activity.
- Knowing which previously used processes contain stopped activities the user might return to and prioritizing those processes more highly to keep them available.
- Helping the app handle having its process killed so the user can return to activities with their previous state restored.
- Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. Example is sharing.

Services

A *service* is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface.

For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it to interact with it.

There are two types of services that tell the system how to manage an app: started services and bound services.

Started services tell the system to keep them running until their work is completed. This might be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music represent different types of started services, which the system handles differently:

- Music playback is something the user is directly aware of, and the app communicates this to the system by indicating that it wants to be in the foreground, with a notification to tell the user that it is running. In this case, the system prioritizes keeping that service's process running, because the user has a bad experience if it goes away.

- A regular background service is not something the user is directly aware of, so the system has more freedom in managing its process. It might let it be killed, restarting the service sometime later, if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. A bound service provides an API to another process, and the system knows there is a dependency between these processes. So if process A is bound to a service in process B, the system knows that it needs to keep process B and its service running for A. Further, if process A is something the user cares about, then it knows to treat process B as something the user also cares about. Because of their flexibility, services are useful building blocks for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they run.

Broadcast receivers

A *broadcast receiver* is a component that lets the system deliver events to the app outside of a regular user flow so the app can respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running.

So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event. Because the alarm is delivered to a Broadcast Receiver in the app, there is no need for the app to remain running until the alarm goes off.

Many broadcasts originate from the system, like a broadcast announcing that the screen is turned off, the battery is low, or a picture is captured. Apps can also initiate broadcasts, such as to let other apps know that some data is downloaded to the device and is available for them to use.

Although broadcast receivers don't display a user interface, they can create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work.

For instance, a broadcast receiver might schedule a JobService to perform some work based on an event using JobScheduler. Broadcast receivers often involve apps interacting with each other, so it's important to be aware of the security implications when setting them up.

Content providers

A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data, if the content provider permits it.

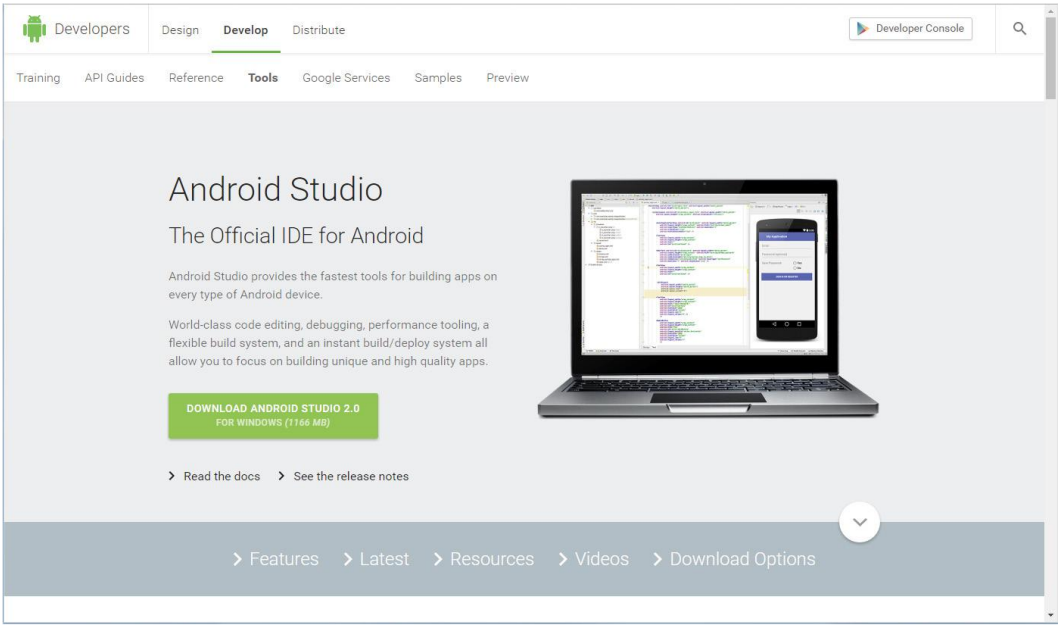
For example, the Android system provides a content provider that manages the user's contact information. Any app with the proper permissions can query the content provider, such as using Contacts. Data, to read and write information about a particular person.

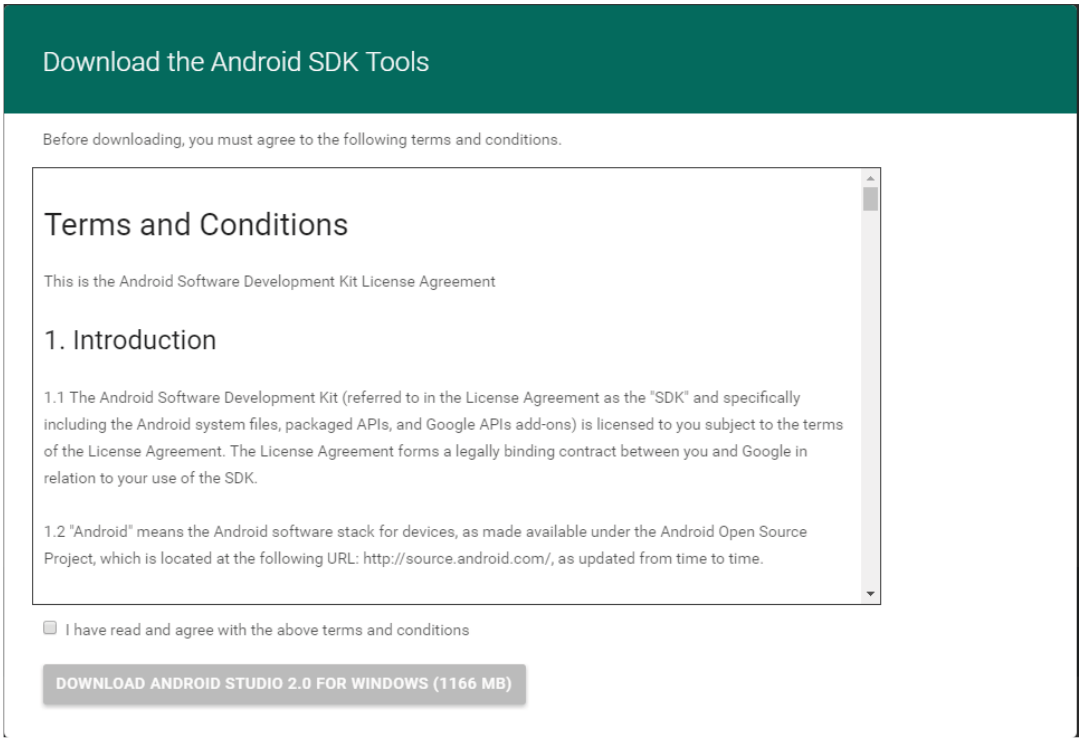
To the system, a content provider is an entry point into an app for publishing named data items, identified by a URI scheme. Thus, an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities which can in turn use them to access the data. Content providers are also useful for reading and writing data that is private to your app and not shared.

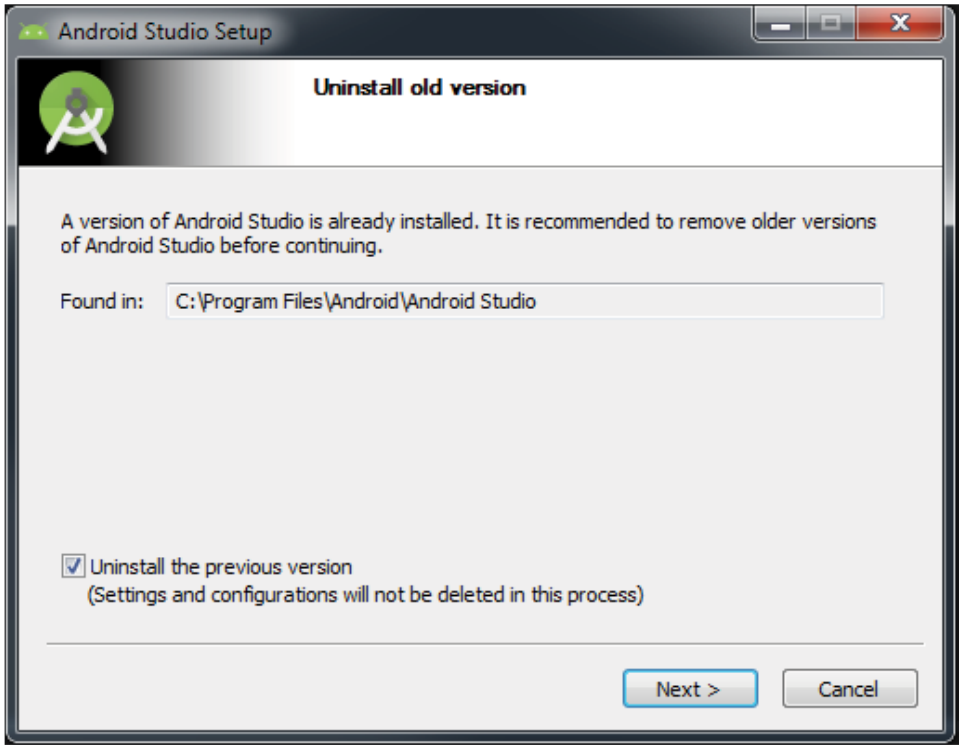
ANDROID STUDIO INSTALLATION:

1. Accept the terms and conditions shown in Figure 1-6.

2. If you have an older version of Android Studio already installed on your computer, the Android Studio Setup prompts you to automatically uninstall it. Even though the old version of Android Studio will be uninstalled, the settings and configurations are retained. You have an opportunity to reapply those settings and configurations to Android Studio after the setup has completed.



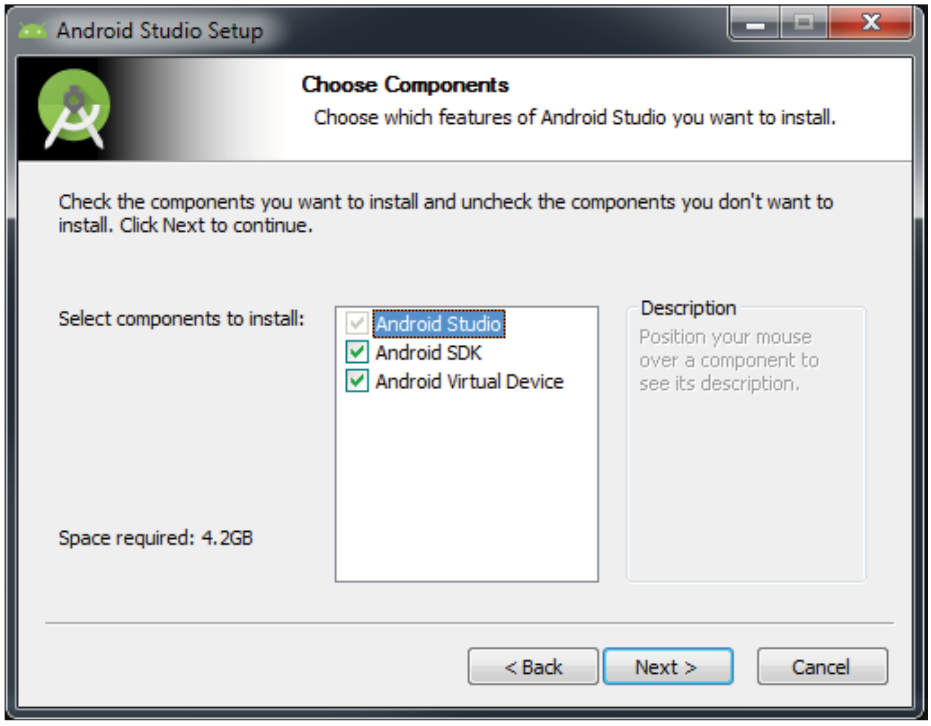




3.Click Next on the Welcome to Android Studio Setup screen

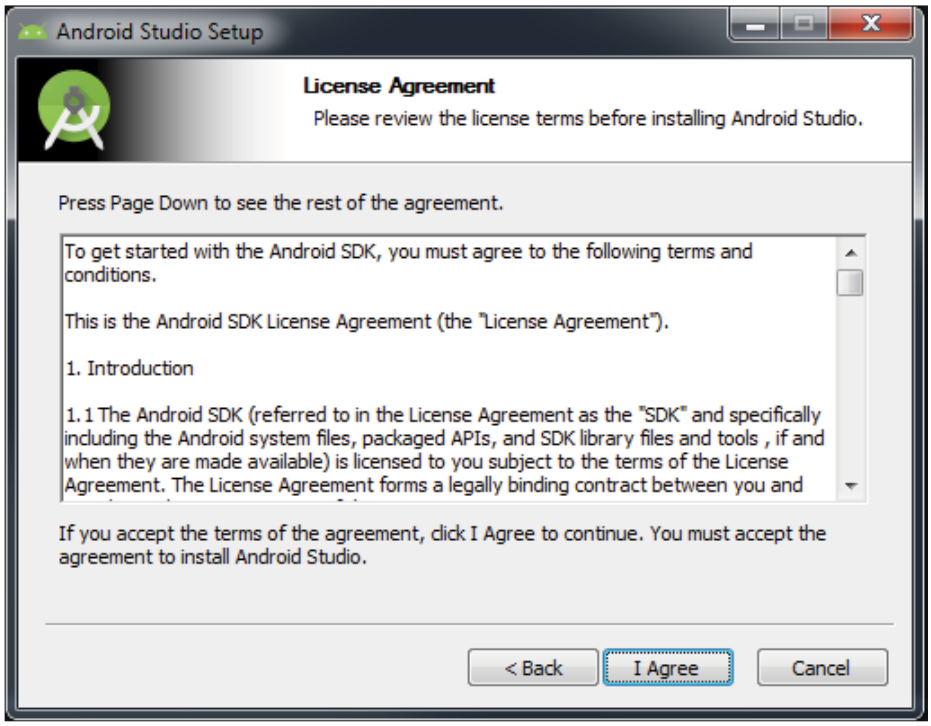
4. Pick which components of Android Studio you want to install. Android Studio is selected by default (and cannot be deselected, Android SDK and Android Virtual Device are also selected by default. Click

Next to accept the default choices and continue.



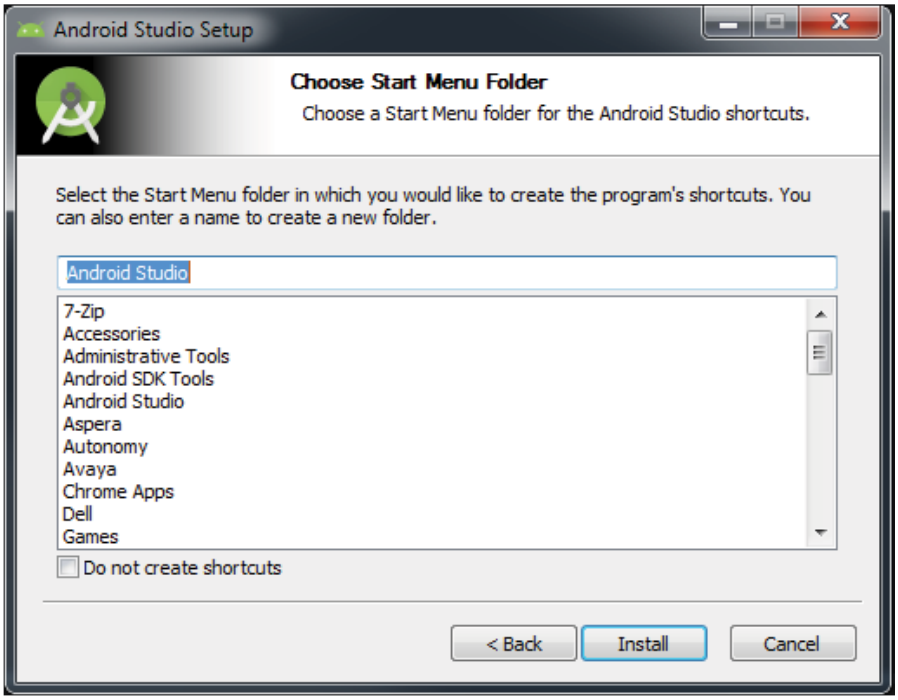
5. You are presented with the License Agreement. Click I Agree

to continue.



6. On the configuration settings screen, it is best to accept the default locations specified

by the setup process and click Next to continue. You see the Choose Start Menu Folder screen . Click Install to kick off the Android Studio 2 installation.



7. Installing Android Studio 2 could take a few minutes, depending on the speed of your computer.

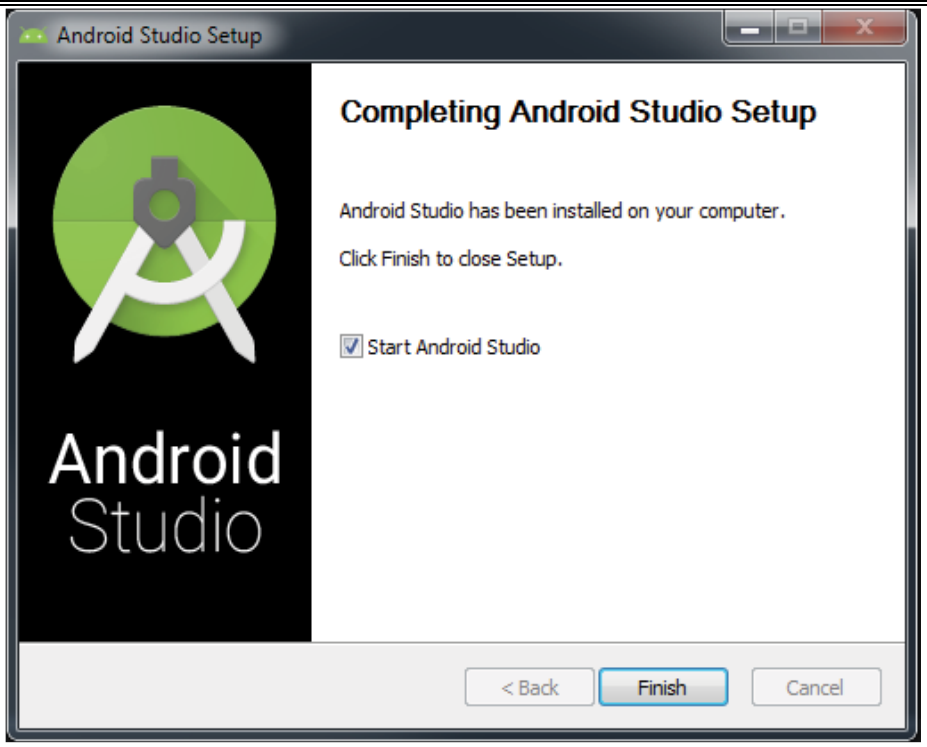
You are presented with a progress bar to help you track the state of the installation.

Android Studio 2 is installed with a default SDK (Software Development Kit), in this case

Marshmallow.

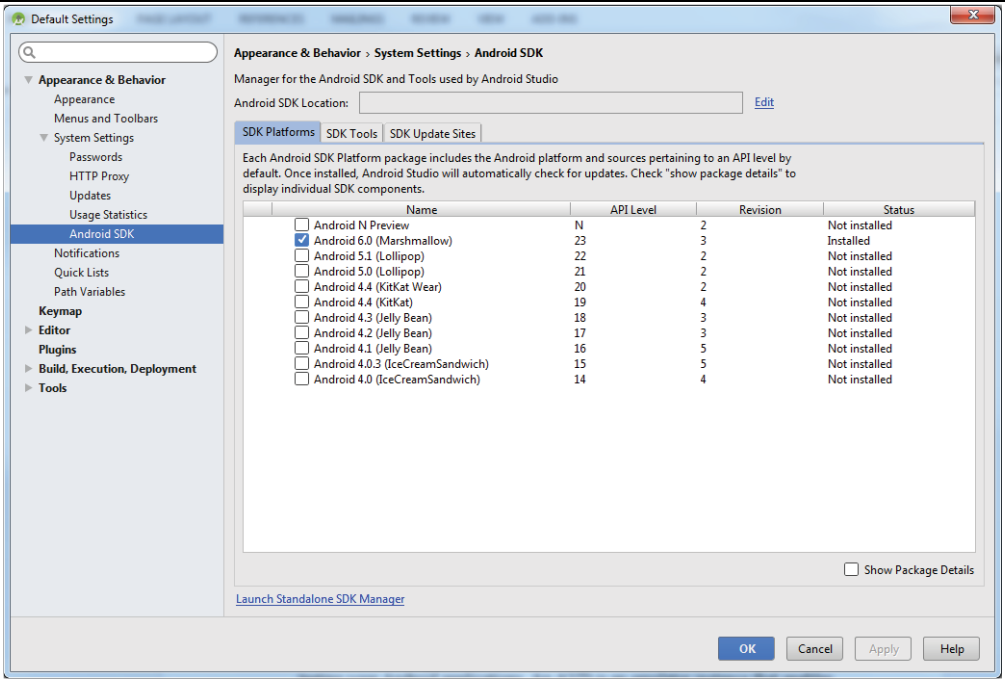
8. When the install is complete, you will see a Completing Android Studio Setup.

Leave the Start Android Studio box checked and click Finish.



Android SDK

- The most important piece of software you need to download is, of course, the Android SDK.
- The Android SDK contains all of the packages and tools required to develop a functional Android application.
- The SDKs are named after the version of Android OS to which they correspond.
- By default, the Marshmallow SDK was installed with Android Studio 2.
- However, if you want to install a different Android SDK, you can do so using the SDK Manager.
- from the Android Studio welcome screen . From this screen, click the Configure drop-down menu in the lower-right corner.
- The Configure selection menu opens. Choose SDK Manager from this menu.
- The SDK configuration screen, shows that the Marshmallow SDK is already installed.



- The setup process for Android Studio is now complete. The next section explains how to set up an Android Virtual Device that you can use to test your applications.

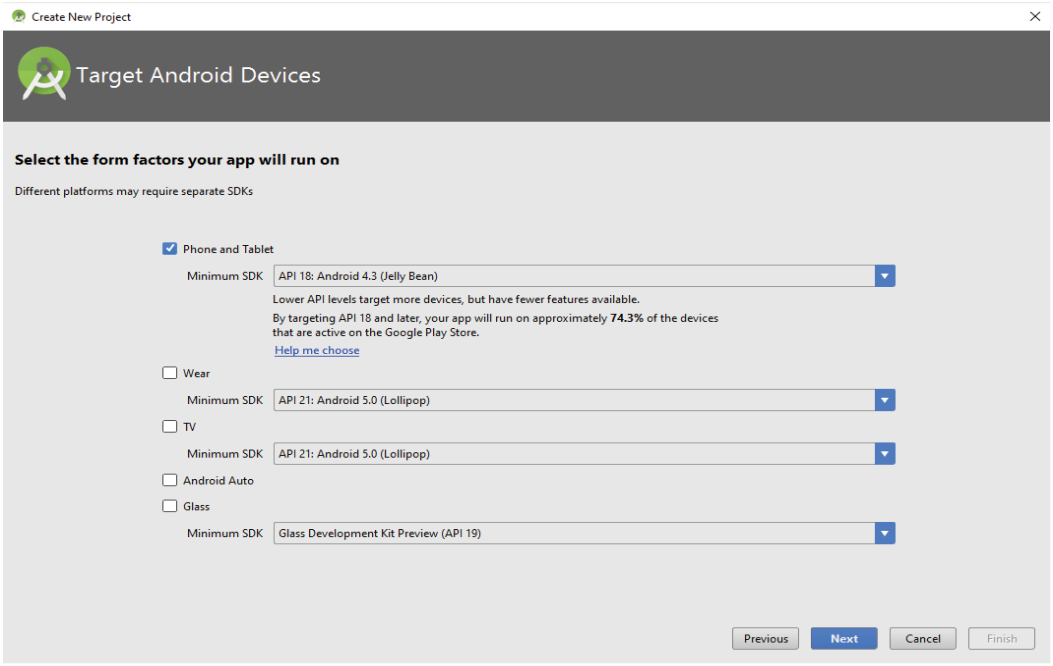
Creating Android Virtual Devices (AVDs)

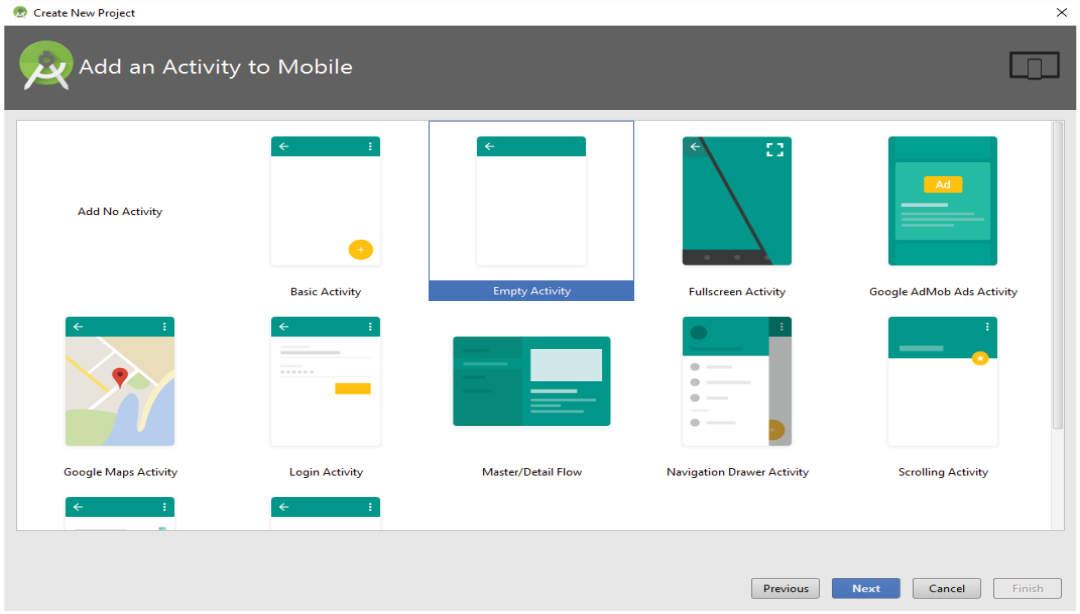
- The next step is to create an Android Virtual Device (AVD) you can use for testing your Android applications.
- An AVD is an emulator instance that enables you to model an actual device. Each AVD consists of a hardware profile; a mapping to a system image; and emulated storage, such as a secure digital (SD) card.
- One important thing to remember about emulators is that they are not perfect.
- There are some applications, such as games (which are GPU heavy) or applications that use sensors such as the GPS or accelerometer. These types of applications cannot be simulated with the same speed or consistency within an emulator as they can when running on an actual device.
- However, the emulator is good for doing some generalized testing of your applications.
- You can create as many AVDs as you want to test your applications with different configurations.

Use the following steps to create an AVD. This example demonstrates creating an AVD (put simply, an Android emulator) that emulates an Android device running Android N on the Nexus 5xhardware specs.

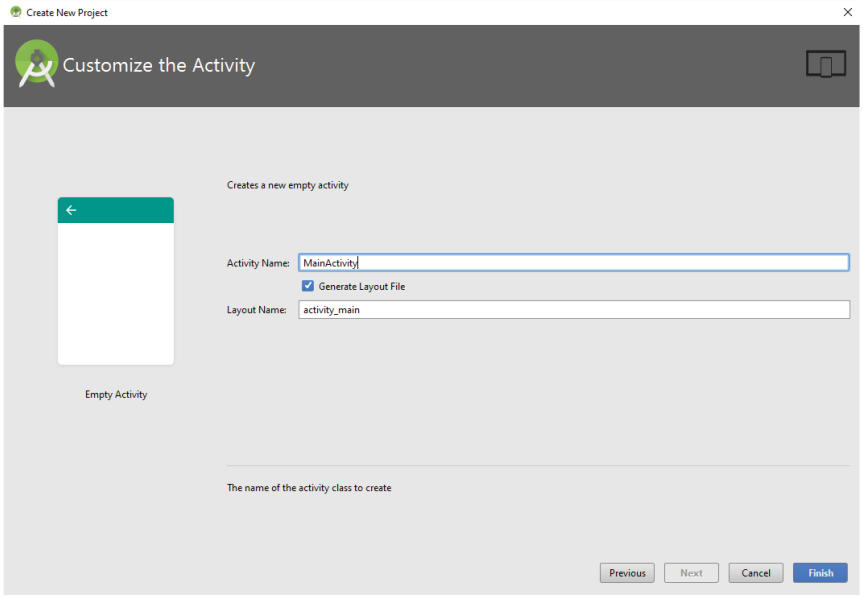
1. Start Android Studio so that the Welcome screen is visible. Click Start a New Android Studio Project. Set up a Hello World project.Type**Chapter1Helloworld** in the Application Name field.

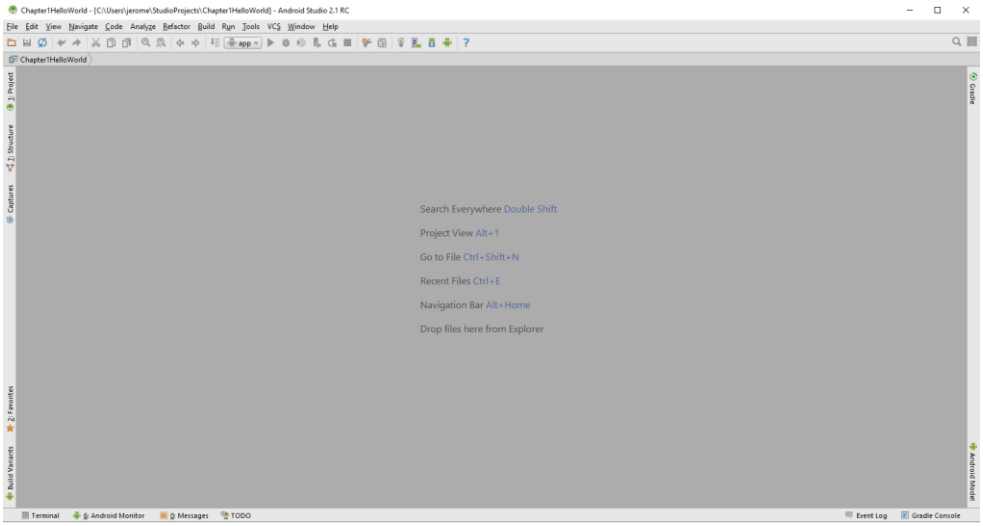
2. You can keep the default values for the other fields on the New Project . Click Next.
3. You should see the Targeted Android Devices screen. By default, the Create New Project
4. Wizard selects for you the Android SDK level that has the greatest activity based on statistics gathered from Google Play. For now, accept the default, and click Next.
5. On the Add an Activity to Mobile screen, accept the default choice—Empty Activity and click Next.



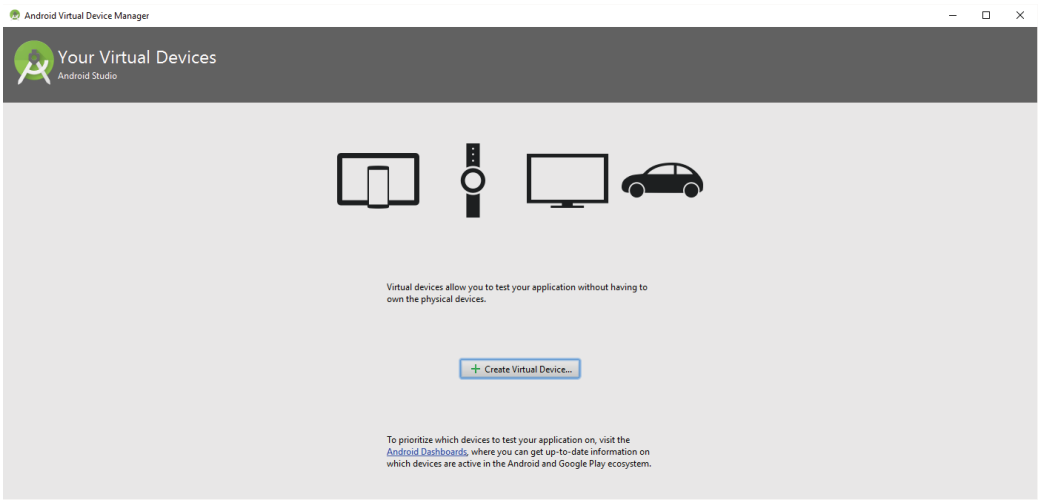


6. Accept all of the defaults on the Customize the Activity screen, and click Finish.





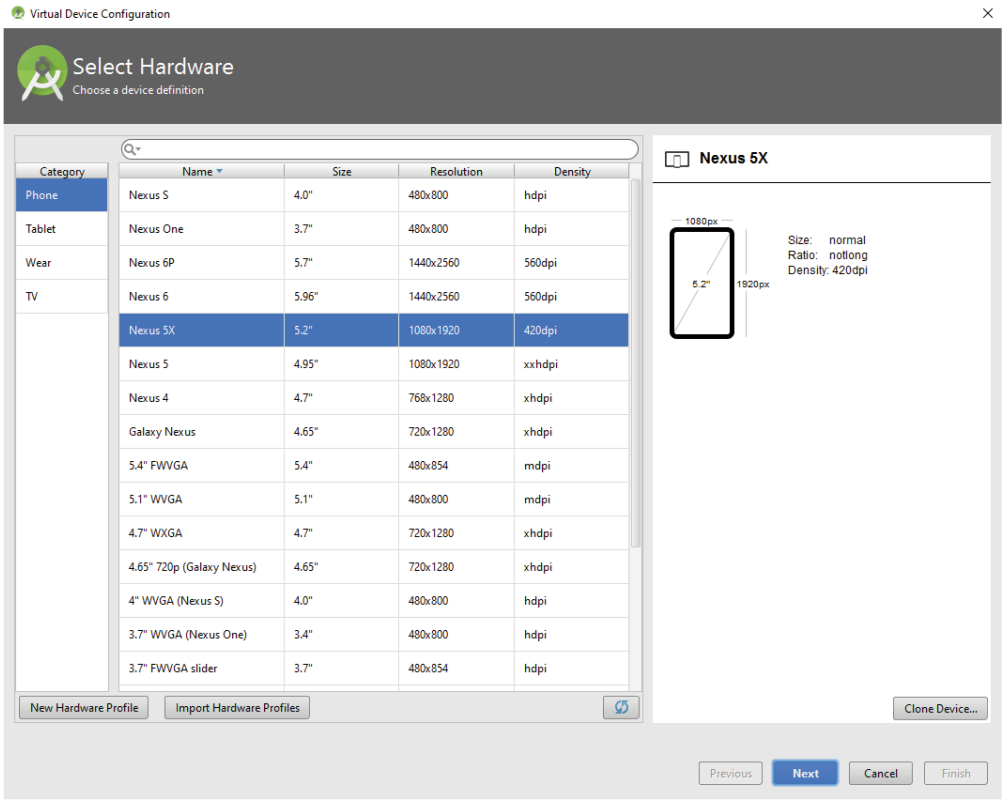
7. Launch the AVD Manager by selecting Tools ➪ Android ➪ AVD Manager or using the AVD Manager button from the toolbar.



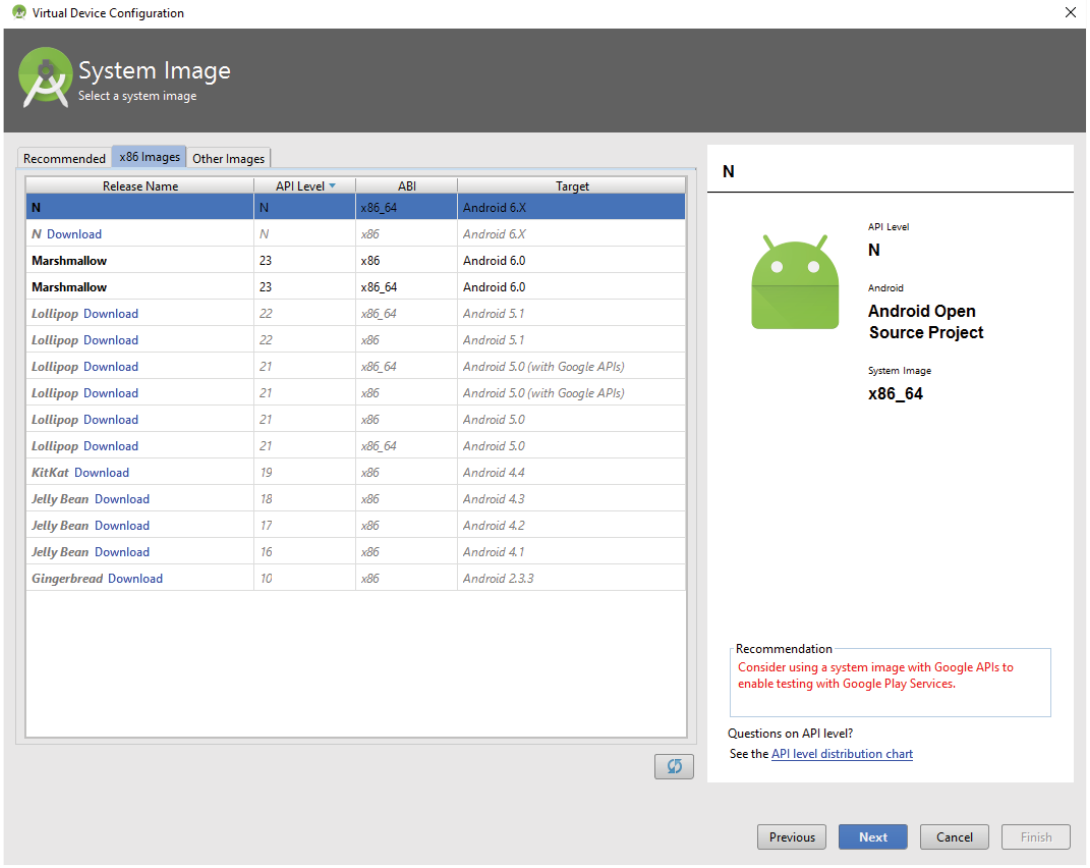
8. Click the + Create Virtual Device button to create a new AVD. The Virtual Device Configuration screen opens.

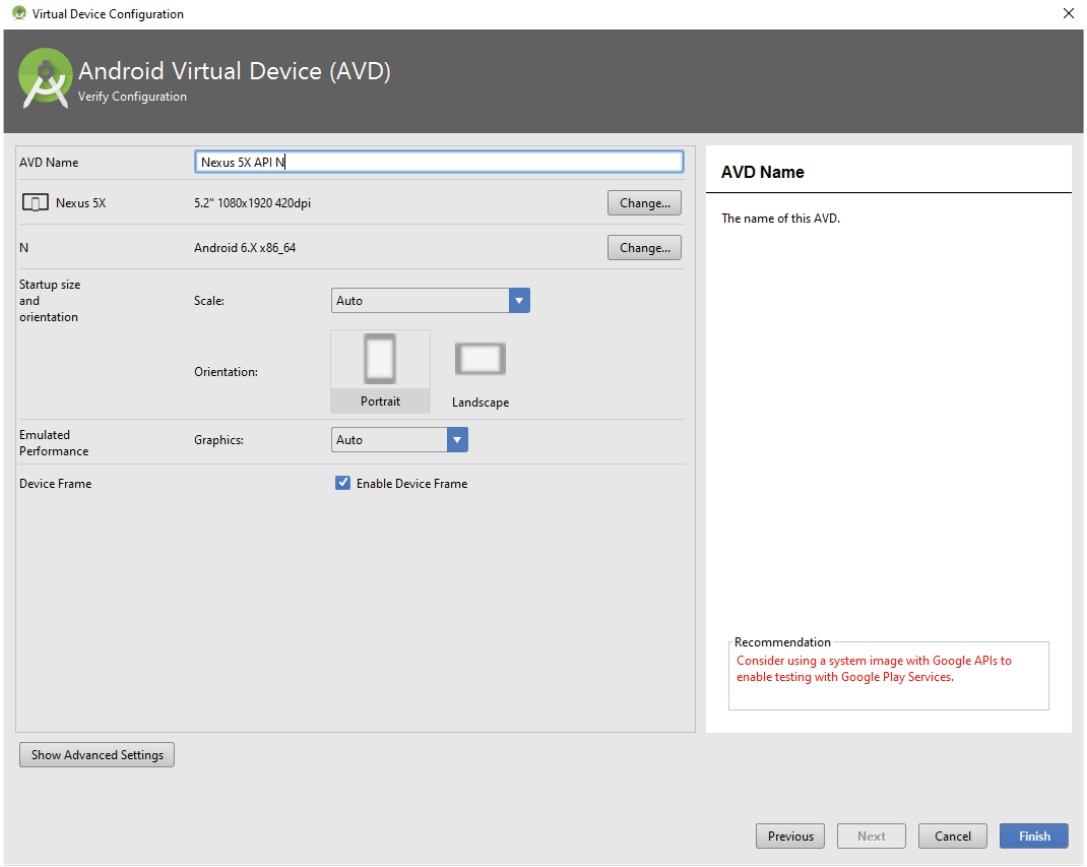
9. Select the Nexus 5x hardware profile and click Next. Although none of the emulators offers the same performance as its actual hardware counterpart, the Nexus 5x should run well on most x86-based desktops, and it still offers some of the mid- to high-end Android device specs.

10. For the system image, select and install the latest option, which at the time this book was written is Android Nougat. Click the x86 Images select N from the



11. In the Android Virtual Device (AVD) dialog, accept the defaults. Click the Finish button to begin building the AVD.





USING THE TEXTVIEW CONTROL

In android UI or input controls are the interactive or View components which are used to design the user interface of an application. In android we have a wide variety of UI or input controls available, those are TextView, EditText ,Buttons,Checkbox,Progressbar,Spinners, etc.

In android, TextView is a user interface control which is used to set and display the text to the user based on our requirements. The TextView control will act as like label control and it won't allow users to edit the text.

A good example of TextView control usage would be to display textual labels for other controls, like "Enter a Date:", "Enter a Name:" or "Enter a Password:".

In android, we can create a TextView control in two ways either in XML layout file or create it in Activity file programmatically.

Specific attributes of TextView controls you will want to be aware of:

- Give the TextView control a unique name using the id property.
- Set the text displayed within the TextView control using the text property; programmatically set with the setText() method.
- Set the layout height and layout width properties of the control as appropriate.
- Set any other attributes you desire to adjust the control's appearance. For example, adjust the text size, color, font or other style settings.
- By default, text contents of a TextView control are left-aligned. However, you can position the text using the gravity attribute. This setting positions your text relative to the control's overall width and height and only really makes sense to use if there is whitespace within the TextView control.
- In XML, this property would appear within your TextView control as: android:gravity="center"
- By default, the background of a TextView control is transparent. That is, whatever is behind the control is shown. However, you can set the background of a control explicitly, to a color resource, or a drawable (picture). In XML, this property would appear within your TextView control as: android:background="#0000ff"
- By default, any text contents within a TextView control is displayed as plain text. However, by setting one simple attribute called autoLink, all you can enable

automatic detection of web, email, phone and address information within the text. In XML, this property would appear within your TextView control as:

android:autoLink="all"

You can control the color of the text within the TextView control by using the textColor attribute. This attribute can be set to a color resource, or a specific color by hex value.

In XML, this property would appear within your TextView control as:

android:textColor="#ff0000"

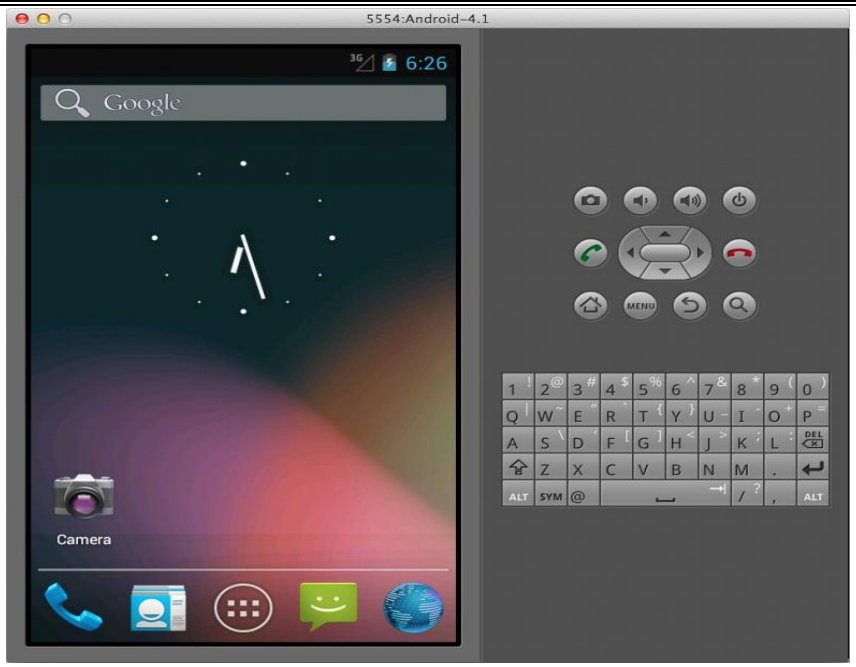
- You can control the style of the text (bold, italic) and font family (sans, serif, monospace) within the TextView control by using the textStyle and typeface attributes. In XML, these properties would appear within your TextView control as: android:textStyle="bold" android:typeface="monospace"

Example:

```
<TextView
android:id="@+id/message"
 android:layout_width="match_parent"
android:layout_height="wrap_content"
tools:context=".HelloWorldAppActivity"
android:typeface="serif"
android:textColor="#0F0"
android:textSize="25dp"
android:textStyle="italic"
android:gravity="center_horizontal" />
```

USING THE ANDROID EMULATOR :

The Android emulator is used for testing and debugging applications before they are loaded onto a real handset. Android emulator is typically used for deploying apps that are developed in your IDE without actually installing it in a device. Android emulators such as Bluestacks can run android apps where in which emulators like AVD and genymotion are used to emulate an entire operating system. The Android emulator is integrated into Eclipse through the ADT plug-in.



Limitations of the Android Emulator

The Android emulator is useful to test Android applications for compatibility with devices of different configurations.

But still, it is a piece of software and not an actual device and has several limitations:

- Emulators no doubt help in knowing how an application may operate within a given environment, but they still don't provide the actual environment to an application. For example, an actual device has memory, CPU, or other physical limitations that an emulator doesn't reveal.
- Emulators just simulate certain handset behavior. Features such as GPS, sensors, battery, power settings, and network connectivity can be easily simulated on a computer.
- SMS messages are also simulated and do not use a real network.
- Phone calls cannot be placed or received but are simulated.
- No support for device-attached headphones is available.
- Peripherals such as camera/video capture are not fully functional.
- No USB or Bluetooth support is available.

The emulator provides some facilities too. You can use the mouse and keyboard to interact with the emulator when it is running. For example, you can use your computer mouse to click, scroll, and drag items on the emulator. You can also use it

to simulate finger touch on the soft keyboard or a physical emulator keyboard. You can use your computer keyboard to input text into UI controls and to execute specific emulator commands. Some of the most commonly used commands are

- Back [ESC button]
- Call [F3]
- End [F4]
- Volume Up [KEYPAD_PLUS, Ctrl-5]
- Volume down [KEYPAD_MINUS, Ctrl-F6]
- Switching orientations [KEYPAD_7, Ctrl-F11/KEYPAD_9, Ctrl-F12]

You can also interact with an emulator from within the DDMS tool. Eclipse IDE provides three perspectives to work with: Java perspective, Debug perspective, and DDMS perspective. The Java perspective is the default and the one with which you have been working up to now. You can switch between perspectives by choosing the appropriate icon in the top-right corner of the Eclipse environment. The three perspectives are as follows:

- The Java perspective—It's the default perspective in Eclipse where you spend most of the time. It shows the panes where you can write code and navigate around the project.

- The Debug perspective—Enables application debugging. You can set breakpoints; step through the code; view LogCat logging information, threads, and so on.

- The Dalvik Debug Monitor Service (DDMS) perspective—Enables you to monitor and manipulate emulator and device status. It also provides screen capture and simulates incoming phone calls, SMS sending, and GPS coordinates. To manage content in the device or emulator, you can use the ADB (Android Debug Bridge).

THE ANDROID DEBUG BRIDGE (ADB)

The Android-Debug-Bridge (abbreviated as adb) is a software-interface for the android system, which can be used to connect an android device with a computer using an USB cable or a wireless connection. It can be used to execute commands on the phone or transfer data between the device and the computer.[1]

The tool is part of the Android SDK(Android Software Development Kit) and is located in the subdirectory platform tools. In previous versions of the SDK it was located in the subdirectory tools.

The Android Debug Bridge is a software interface between the device and the local computer, which allows the direct communication of both components. This includes the possibility to transfer files from one component to the other one, as well as executing commands from the computer on the connected device. The ADB can be used through a command line windows, terminal/shell in Linux-based systems, a command line (cmd) for Windows. The main advantage is to execute commands on the phone directly out of the computer, without any direct user interaction to the phone, which makes especially debugging a lot easier.

It is a client-server program that includes three components:

- A client, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.
- A daemon (adbd), which runs commands on a device. The daemon runs as a background process on each device.
- A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

When you start an adb client, the client first checks whether there is an adb server process already running. If there isn't, it starts the server process. When the server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients—all adb clients use port 5037 to communicate with the adb server.

The server then sets up connections to all running devices. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, the range used by the first 16 emulators. Where the server finds an adb daemon (adbd), it sets up a connection to that port. Note that each emulator uses a pair of sequential ports — an even-numbered port for console connections and an odd-numbered port for adb connections.

Once the server has set up connections to all devices, you can use adb commands to access those devices. Because the server manages connections to devices and handles commands from multiple adb clients, you can control any device from any client.

Android Studio Project Structure

The android project contains different types of app modules, source code files, and resource files.

1. Manifests Folder
2. Java Folder
3. res (Resources) Folder
   - Drawable Folder
   - Layout Folder
   - Mipmap Folder
   - Values Folder
4. Gradle Scripts

Manifests Folder

Manifests folder contains AndroidManifest.xml for creating our android application. This file contains information about our application such as the Android version, metadata, states package for java file, and other application components. It acts as an intermediator between android OS and our application.

Java folder

The Java folder contains all the java and Kotlin source code (.java) files that we create during the app development, including other Test files. If we create any new project using java, by default the class file MainActivity.java file will create automatically under the package name

Resource (res) folder

The resource folder is the most important folder because it contains all the non-code sources like images, XML layouts, and UI strings for our android application.

res/drawable folder

It contains the different types of images used for the development of the application. We need to add all the images in a drawable folder for the application development.

res/layout folder

The layout folder contains all XML layout files which we used to define the user interface of our application. It contains the **activity_main.xml** file.

res/mipmap folder

This folder contains launcher.xml files to define icons that are used to show on the home screen. It contains different density types of icons depending upon the size of the device such as hdpi, mdpi, xhdpi.

res/values folder

Values folder contains a number of XML files like strings, dimensions, colors, and style definitions. One of the most important files is the **strings.xml** file which contains the resources.

Gradle Scripts folder

Gradle scripts: With Android studio, Google switched to the new advanced building system, Gradle. It is a JVM based build system. If you want to make the package building task automatically, then you can write your own script in

java or groovy and distribute it. Gradle allows to create different variants apk files for the same application project.

Build system and Gradle :

The build system is responsible to build, test an android system and also prepare the deployable files for the specified platform. In simple words, build system generates the .apk files for the application project.

With Android Studio, the advanced build system allows the developers to configure the build systems manually, create different variant APK files from single project (without modifying the execution code) and share the code and resources from other modules. Before Android Studio, Eclipse was the IDE used for android development. Eclipse kept all the .java files (in src directory) and resource files (in res directory) in the same directory. That allows the build system to group all the files into an intermediate code and finally generates .apk file.

Android Studio uses Gradle as its build system. One intriguing feature that Gradle offers is that it allows you to write your own script to automate the task of building an app. As Gradle is plug-in based system, if you have your own programming language then you can write the plug-in in the script using java or groovy and share.

Activities and intents:

An activity is a window that contains the user interface of your applications. An application can have zero or more activities. Typically, applications have one or more activities, and the main aim of an activity is to interact with the user. From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's life cycle.

To create an activity, you create a Java class that extends the Activity base class:

```
import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
 /** Called when the activity is first created. */
 @Override
```

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main); }}

Your activity class would then load its UI component using the XML file defined in your res/layout

folder. In this example, you would load the UI from the main.xml file:

setContentView(R.layout.main);

Every activity you have in your application must be declared in your AndroidManifest.xml file:

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

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

package=”net.learn2develop.Activities”

android:versionCode=”1”

android:versionName=”1.0”>

<application android:icon=”@drawable/icon”

android:label=”@string/app_name”>

<activity android:name=”.MainActivity”

android:label=”@string/app_name”>

<intent-filter>

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

<category

android:name=”android.intent.category.LAUNCHER” />

</intent-filter>

</activity>

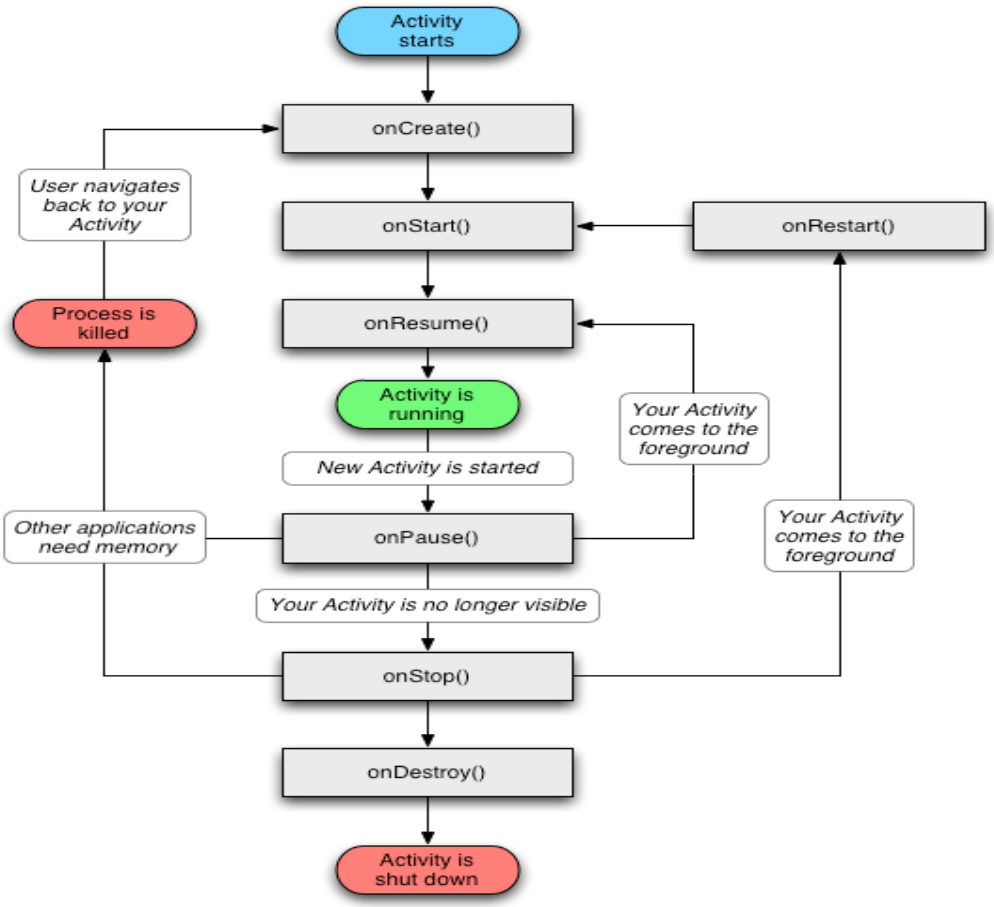
</application>

<uses-sdk android:minSdkVersion=”9” />

</manifest>

The life cycle of an activity:

The Activity base class defines a series of events that governs the life cycle of an activity.



The Activity class defines the following events:

onCreate() — Called when the activity is first created

onStart() — Called when the activity becomes visible to the user

onResume() — Called when the activity starts interacting with the user

onPause() — Called when the current activity is being paused and the previous activity is

being resumed

onStop() — Called when the activity is no longer visible to the user

onDestroy() — Called before the activity is destroyed by the system (either manually or by

the system to conserve memory)

onRestart() — Called when the activity has been stopped and is restarting again.

By default, the activity created for you contains the onCreate() event. Within thi event handler is the code that helps to display the UI elements of your screen.

- onCreate () is called when your Activity is getting created for the first time. It is called only once during the entire Activity Lifecycle. One of the important things you are supposed to do is to set the Activity Layout through setContentView function. Also, you can use onCreate to initialize your variables. In any Android application, whenever you create an Activity, the minimum method which you need to override is onCreate().

```
class MainActivity extends Activity
@Override
protected void onCreate (Bundle savedinstanceState)
super.onCreate(savedlinstanceState);
setContentView(R.layout.activity_main);
```

- onStart():onStart gets called just before the Activity becomes visible to the user.onStart is called from two places - after onRestart and OnCreate. onStart is always followed by OnResume or OnStop. You can use onstart to reset Activity data, reinitialize variables etc.
- onResume(): onResume gets called when your Activity comes into the foreground, and it becomes visible to the user. Af this point, the Activity is on top of the Activity stack, and the user can start interacting with the Activity.onResume is typically used to register Listeners, bind to Services etc. onResume is a good place to refresh your Ul with any new changes which might have occurred during the period in which the Activity was nof visible. For example, if you are polling a Service in the background (like checking for new tweets), onResume is a good place to update your screen with new results.
- onPause(): onPause is called when another android activity comes on top of your Activity. Typically anything that steals your user away from your Activity will result in onPause.

ln OnPause, we either save the application data, or stop background threads etc.

It is always guaranteed that whenever your Activity is becoming invisible or partially invisible, onP ause will be called. But once onPause is called, Android reserves the right to kill your Activity at any point

- onStop(): onstop is called when your Activity is no longer visible to the user, it is similar to onPause but here you will not see your android activity entirely. Typically whenever you see a dialog box which requires your attention like battery low, network connection your current android activity becomes partially visible and popup box comes on the top. This is the point where only onPause will be called.

- onRestart():It is similar to onCreate, but onRestart gets called only after onStop. This is the method which you can use to know if your application is starting fresh or getting restarted .In onRestart, you will get your application to save the state and reinitialize all the variables.

- OnDestroy(): This is the method which will be called when your Activity is getting killed. This is the final call the Activity will receive in its Lifecycle . When the user press back button on any Activity the foreground activity gets destroyed and control will return to the previous Activity. There is no guaranty that onDestroy will be called. Only when the system is low on resources or user press the back button or if you use finish () explicitly in your code, onDestroy gets called on Destroy is there to let your app have the fine chance to clean things up before the Activity cease to exist.

Intent:

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

Starting an activity

An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data.

```
public class MainActivity extends AppCompatActivity {
Button b;
TextView t;
EditText e;
Button b2;


@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
b=(Button) findViewById(R.id.b);
e=(EditText) findViewById(R.id.e);
t=(TextView) findViewById(R.id.text1);
b2=(Button)findViewById(R.id.start_act);
b.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
t.setText(e.getText().toString());
            }
        });
    }
public void start_activity(View v)

    {
Intent i=new Intent(MainActivity.this,MainActivity2.class);
       startActivity(i);

    }

}
```

MainActivity2.java:

```
public class MainActivity2 extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main2);

    }
}
```

activity_main.xml

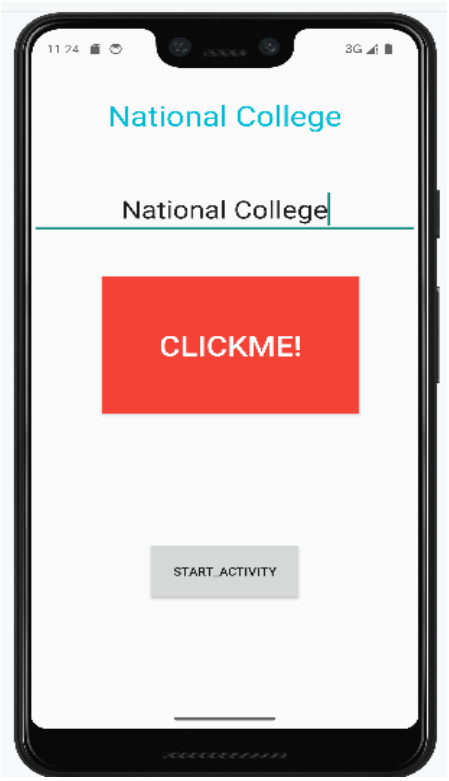
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"

tools:context=".MainActivity">


<TextView
android:id="@+id/text1"
android:layout_width="match_parent"
android:layout_height="189dp"
android:gravity="center"
android:hint="Name"
android:textColor="#00BCD4"
android:textSize="34sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.095" />

<EditText
android:id="@+id/e"
android:layout_width="match_parent"
android:layout_height="62dp"
android:ems="10"
android:gravity="center"
android:hint="Enter Your Name"
android:inputType="text"
android:textSize="60sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/text1"
app:layout_constraintVertical_bias="0.081" />

<Button
```

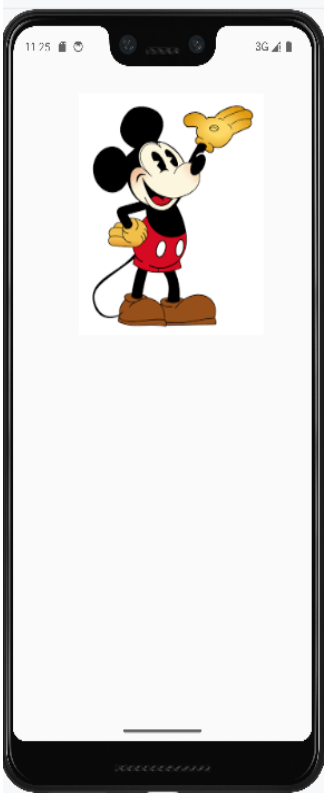
```
android:id="@+id/b"
android:layout_width="273dp"
android:layout_height="166dp"
android:background="#F44336"
android:text="ClickMe!"
android:textColor="@color/white"
android:textSize="34sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.546"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/e"
app:layout_constraintVertical_bias="0.116" />

<Button
android:id="@+id/start_act"
android:layout_width="165dp"
android:layout_height="74dp"
android:text="Start_Activity"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
android:onClick="start_activity"
app:layout_constraintTop_toBottomOf="@+id/b" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

Activity_main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity2">

<ImageView
android:id="@+id/imageView"
android:layout_width="356dp"
android:layout_height="286dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.709"
app:layout_constraintStart_toStartOf="parent"
```

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.143"
app:srcCompat="@drawable/img_1" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

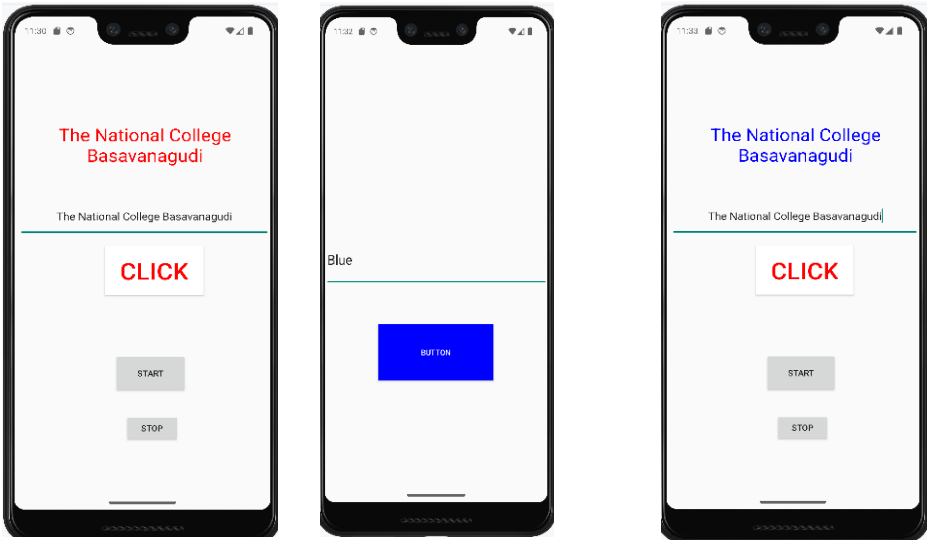




If you want to receive a result from the activity when it finishes, call startActivityForResult(). Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback.

```
public class MainActivity extends AppCompatActivity {
Button b;
EditText e;
TextView t;
Button b2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        b=(Button) findViewById(R.id.b);
        e=(EditText) findViewById(R.id.e);
        t=(TextView) findViewById(R.id.t);
```

```
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                t.setText(e.getText().toString());
                Intent i=new Intent(MainActivity.this,MainActivity2.class);
                startActivityForResult(i,1);

            }
        });

        }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, @Nullable

Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if(requestCode==1)
        {
          t.setTextColor(Color.parseColor( data.getStringExtra("MSG".toString())));
        }
    }
}



public class MainActivity2 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main2);
        Button b=(Button) findViewById(R.id.button);
        EditText e=(EditText) findViewById(R.id.editTextText);
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i=new Intent();
                i.putExtra("MSG",e.getText().toString());
                setResult(1,i);
                finish();

            }       });    }}
```



Starting a service

A Service is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with JobScheduler. For more information about JobScheduler, see its API-reference documentation.

For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the Service class. You can start a service to perform a one-time operation (such as downloading a file) by passing an Intent to startService(). The Intent describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService(). For more information, see the Services guide.

```
public class MainActivity extends AppCompatActivity {
Button b2;
Button b3;

@SuppressLint("MissingInflatedId")
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
b=(Button) findViewById(R.id.b);
b2=(Button)findViewById(id.start_service) ;
b3=(Button)findViewById(id.stop_service);
b2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i=new Intent(MainActivity.this,MyService.class);
                startService(i);
```

```
        }
    });
b3.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i=new Intent(MainActivity.this,MyService.class);
            stopService(i);


        }
    });
}}
```

MyService.java:

```
public class MyService extends Service {
// declaring object of MediaPlayer
private MediaPlayer player;

@Override

// execution of service will start
   // on calling this method
public int onStartCommand(Intent intent, int flags, int startId) {


player = MediaPlayer.create( this, Settings.System.DEFAULT_RINGTONE_URI );
player.setLooping( true );
player.start();

// returns the status of the program
return START_STICKY;
   }

@Override

// execution of the service will  stop on calling this method
public void onDestroy() {
super.onDestroy();
// stopping the process
player.stop();
   }


@Override
```

```
public IBinder onBind(Intent intent) {
return null;
    }
}
```

- **Delivering a broadcast**

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast().

MainActivity.java:

```
public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
IntentFilter i = new IntentFilter("android.intent.action.BATTERY_LOW");
MyReceiver m=new MyReceiver();
        registerReceiver(m,i);

    }
}
```

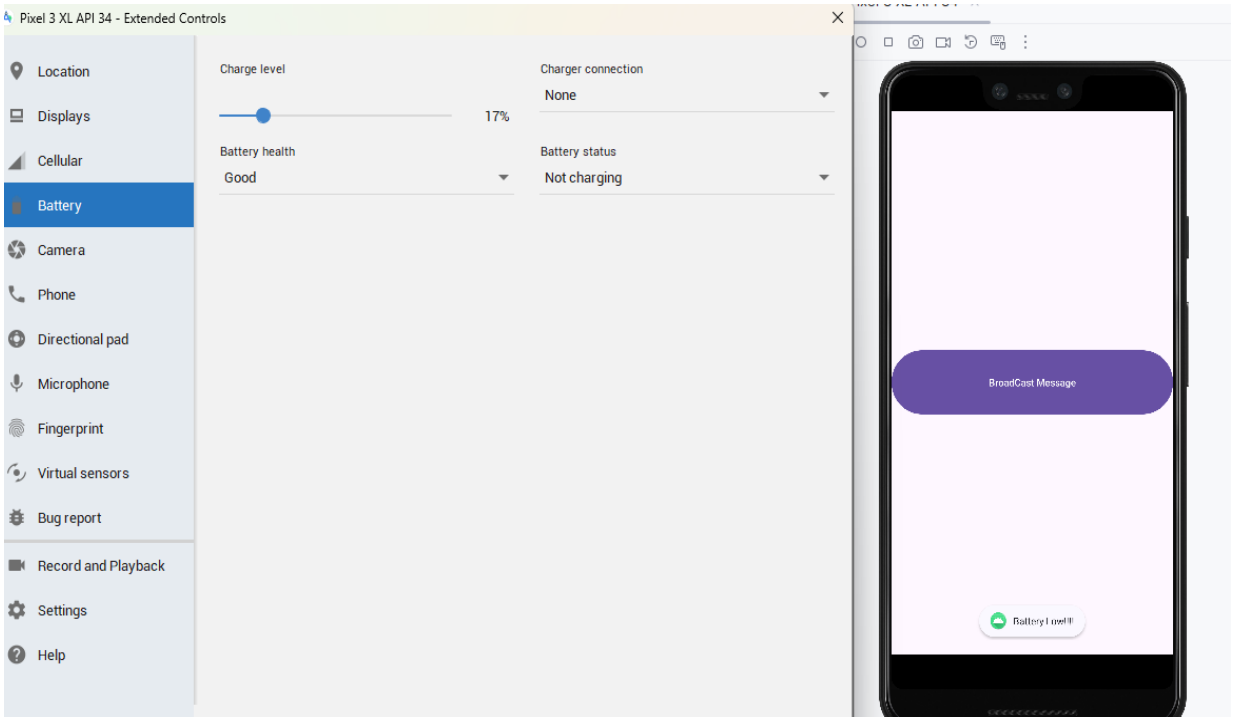
MyReceiver.java:

```
public class MyReceiver extends BroadcastReceiver {

@Override
public void onReceive(Context context, Intent intent) {

Toast.makeText(context,"Battery Low!!!!",Toast.LENGTH_LONG).show();
    }
}
```

Add the below mentioned code in AndriodManifest.xml

```
<receiver
android:name=".MyReceiver"
android:enabled="true"
android:exported="true" >
<intent-filter>
<action android:name="android.intent.action.BATTERY_LOW">

</action>
```

</intent-filter>
</receiver>

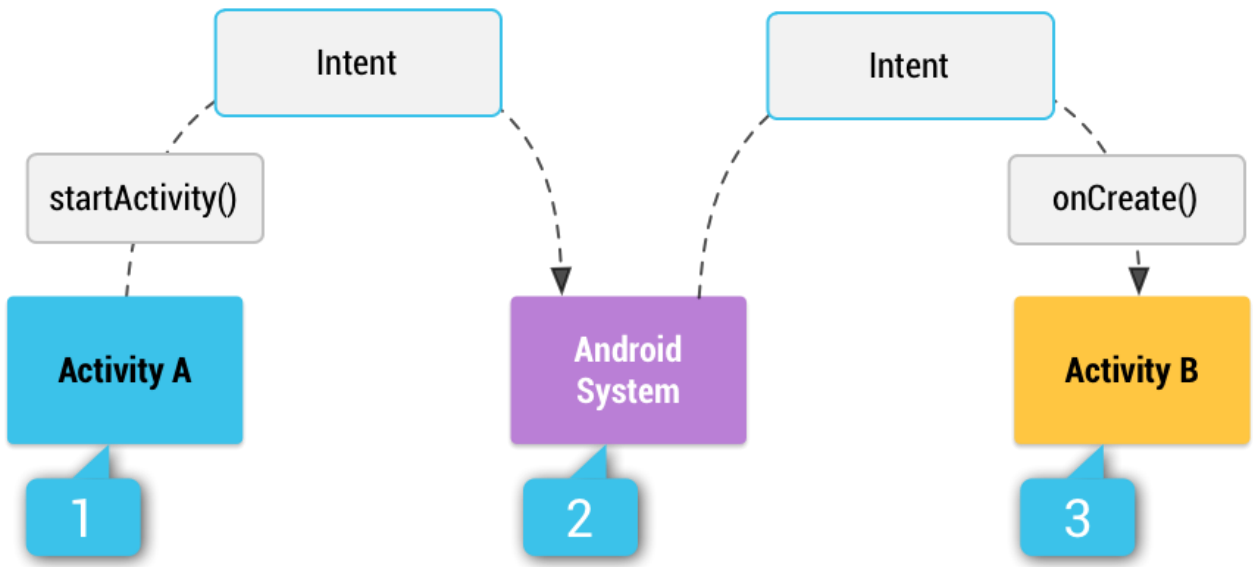


INTENT TYPES

There are two types of intents:

- **Explicit intents**
- **Implicit intents**

The below figure shows how an intent is used when starting an activity. When the Intent object names a specific activity component explicitly, the system immediately starts that component.



An implicit intent is delivered through the system to start another activity:

Activity A creates an Intent with an action description and passes it to startActivity().

The Android System searches all apps for an intent filter that matches the intent When a match is found.

The system starts the matching activity (*Activity B*) by invoking its onCreate() method and passing it the Intent.

When you use an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent.

BUILDING AN INTENT

An Intent object carries information that the Android system uses to determine which component to start (such as the exact component name or component category that should receive the intent), plus information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon).

The primary information contained in an Intent is the following:

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). If you need to start a specific component in your app, you should specify the component name.

This field of the Intent is a ComponentName object, which you can specify using a fully qualified class name of the target component, including the package name of

the app, for example, com.example.ExampleActivity. You can set the component name with setComponent(), setClass(), setClassName(), or with the Intent constructor.

Action

A string that specifies the generic action to perform (such as *view* or *pick*).In the case of a broadcast intent, this is the action that took place and is being reported. The action largely determines how the rest of the intent is structured—particularly the information that is contained in the data and extras.You can specify your own actions for use by intents within your app (or for use by other apps to invoke components in your app), but you usually specify action constants defined by the Intent class or other framework classes. Here are some common actions for starting an activity:

ACTION_VIEW

Use this action in an intent with startActivity() when you have 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, you should use this in an intent with startActivity() when you have some data that the user can share through another app, such as an email app or social sharing app.

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.

When creating an intent, it's often important to specify the type of data (its MIME type) in addition to its URI. For example, an activity that's able to display images probably won't be able to play an audio file, even though the URI formats could be similar. Specifying the MIME type of your data helps the Android system find the best component to receive your intent. However, the MIME type can sometimes be inferred from the URI—particularly when the data is a content: URI. A content: URI indicates the data is located on the device and controlled by a ContentProvider, which makes the data MIME type visible to the system.
To set only the data URI, call setData(). To set only the MIME type, call setType().

Category:

Uses the android:name attribute to specify under which circumstances the actio should be serviced. Each Intent Filter tag can include multiple category tags.

You can specify your own categories or use the following standard values provide by Android:

ALTERNATIVE: This category specifies that this action should be available as a alternative to the default action performed on an item of this data type. For example where the default action for a contact is to view it, the alternative could be to edit it.

SELECTED_ALTERNATIVE: Similar to the ALTERNATIVE category, bu whereas that category will always resolve to a single action using the intent resolutio described next, SELECTED_ALTERNATIVE is used when a list of possibilities i required..

BROWSABLE: Specifies an action available from within the browser. When a Intent is fired from within the browser, it will always include the browsable category If you want your application to respond to actions triggered within the browser (e.g. intercepting links to a particular website), you must include the browsable category.

DEFAULT: Set this to make a component the default action for the data typ specified in the Intent Filter. This is also necessary for Activities that are launche using an explicit Intent.

HOME : By setting an Intent Filter category as home without specifying an action you are presenting it as an alternative to the native home screen.

LAUNCHER :Using this category makes an Activity appear in the applicatio launcher.

Intent Filters Specifies the types of intents that an activity, service, or broadcas receiver can respond to. An intent filter declares the capabilities of its paren component: what an activity or service can do and what types of broadcasts a receive can handle.

It opens the component to receiving intents of the advertised type while filtering ou those that aren't meaningful for the component. Most of the contents of the filter ar described by its <action>, <category>, and <data>sub elements.

Attributes of Intent Filters are:

android:icon: An icon that represents the parent activity, service, or broadcas receiver when that component is presented to the user as having the capabilit described by the filter.

android:label: A user-readable label for the parent component. This label, rathe than the one set by the parent component, is used when the component is presented t the user as having the capability described by the filter.

android:priority: The priority given to the parent component with regard t handling intents of the type described by the filter. This attribute has meaning fo both activities and broadcast receivers.

android:order: The order in which the filter is processed when multiple filter match.

android:autoVerify: Whether Android needs to verify that the Digital Asset Link JSON file from the specified host matches this application, default value is false.

The following snippet shows an Intent Filter for an Activity that can perform the SHOW_DAMAGE action as either a primary or an alternative action based on it mime type.

<intent-filter>

<action android:name=”com.paad.earthquake.intent.action.SHOW_DAMAGE” />

<category android:name=”android.intent.category.DEFAULT”/>

<category android:name=”android.intent.category.SELECTED_ALTERNATIVE”/>

<data android:mimeType=”vnd.earthquake.cursor.item/*”/>

</intent-filter>

These properties listed above (component name, action, data, and category) represent the defining characteristics of an intent. By reading these properties, the Android system is able to resolve which app component it should start. However, an intent can carry additional information that does not affect how it is resolved to an app component. An intent can also supply the following information:

Extras

Key-value pairs that carry additional information required to accomplish the requested action. Just as some actions use particular kinds of data URIs, some actions also use particular extras.

You can add extra data with various putExtra() methods, each accepting two parameters: the key name and the value. You can also create a Bundle object with all the extra data, then insert the Bundle in the Intent with putExtras().

For example, when creating an intent to send an email with ACTION_SEND, you can specify the to recipient with the EXTRA_EMAIL key, and specify the subject with the EXTRA_SUBJECT key.

Example:

MainActivity2.java:

```
public class MainActivity extends AppCompatActivity {
Random r =new Random();
Calendar c = Calendar.getInstance();
int x,y,res;
TextView t1,t2,t3;
EditText e;
Button b;
```

```
int count=0;



@SuppressLint("MissingInflatedId")
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
x = r.nextInt(100);
y = r.nextInt(100);
t1 = (TextView) findViewById(R.id.n1);
t2 = (TextView) findViewById(R.id.n2);
t3 = (TextView) findViewById(R.id.disp);
e = (EditText) findViewById(R.id.editTextText);
b = (Button) findViewById(R.id.button);
t1.setText((String.valueOf(x)));
t2.setText(String.valueOf(y));
b.setOnClickListener(new View.OnClickListener() {
@Override


public void onClick(View v) {
if (Integer.parseInt(e.getText().toString()) == (x + y))
                {
count++;
t3.setText("Correct!!");
t3.setTextColor(Color.parseColor("Green"));
x = r.nextInt(100);
y = r.nextInt(100);
t1.setText((String.valueOf(x)));
t2.setText(String.valueOf(y));
e.getText().clear();


                } else
{
t3.setText("Wrong!!");
t3.setTextColor(Color.parseColor("Red"));
Intent i=new Intent(MainActivity.this, MainActivity2.class);
i.putExtra("COUNT",String.valueOf(count));
                    startActivity(i);

                }
```

```
            }

        });
    }
    }
```

MainActivity2.java:

```
public class MainActivity2 extends AppCompatActivity {
TextView t;
Button b;
@SuppressLint("MissingInflatedId")
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main2);
t=(TextView)findViewById(R.id.textView);
Intent i=getIntent();
String s= i.getStringExtra("COUNT");
t.setText("Your Total Correct Ans is:"+s);
b=(Button) findViewById(R.id.button2);
b.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i=new Intent(MainActivity2.this,MainActivity.class);
            startActivity(i);
          }
      });
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<TextView
```

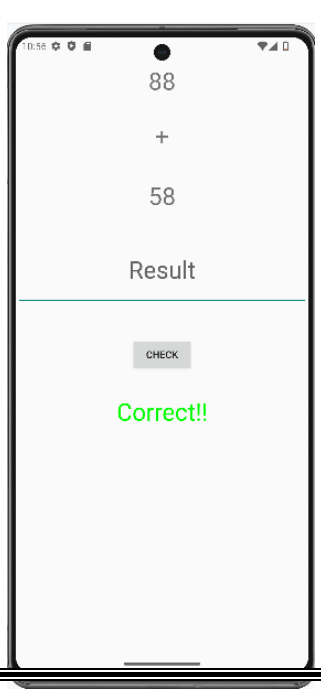
```
android:id="@+id/textView3"
android:layout_width="match_parent"
android:layout_height="66dp"
android:gravity="center"
android:text="+"
android:textSize="34sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.127" />

<TextView
android:id="@+id/n1"
android:layout_width="match_parent"
android:layout_height="89dp"
android:layout_marginTop="16dp"
android:gravity="center"
android:textSize="34sp"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />

<TextView
android:id="@+id/n2"
android:layout_width="match_parent"
android:layout_height="86dp"
android:layout_marginTop="8dp"
android:layout_marginBottom="16dp"
android:gravity="center"
android:textSize="34sp"
app:barrierMargin="@dimen/material_emphasis_high_type"
app:layout_constraintBottom_toTopOf="@+id/editTextText"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/textView3"
app:layout_constraintVertical_bias="0.0"
app:layout_marginBaseline="20dp" />

<EditText
android:id="@+id/editTextText"
android:layout_width="match_parent"
android:layout_height="102dp"
android:layout_marginBottom="44dp"
```

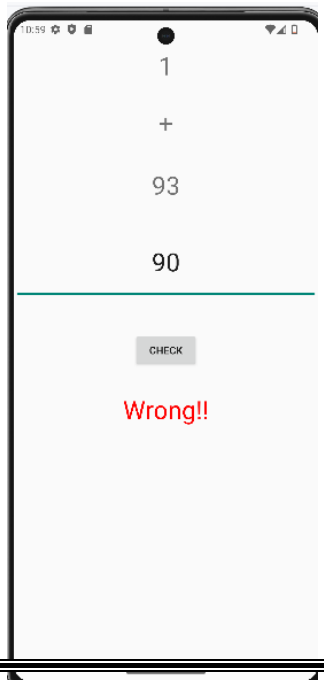
```
android:ems="10"
android:gravity="center"
android:hint="Result"
android:inputType="text"
android:textSize="34sp"
app:layout_constraintBottom_toTopOf="@+id/button"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_marginBaseline="20dp" />

<Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Check"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.499" />

<TextView
android:id="@+id/disp"
android:layout_width="343dp"
android:layout_height="80dp"
android:layout_marginTop="16dp"
android:layout_marginBottom="16dp"
android:gravity="center"
android:textSize="34sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/button"
app:layout_constraintVertical_bias="0.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

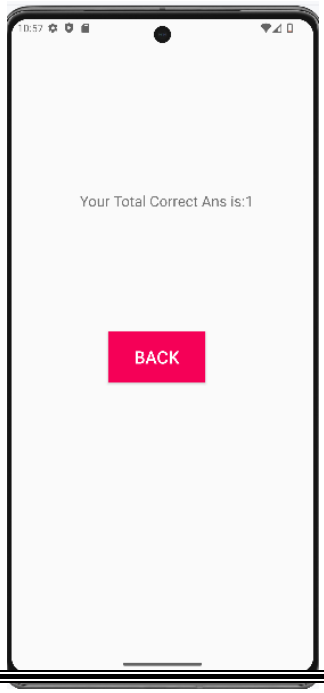
activity_main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

```
tools:context=".MainActivity2">

<TextView
android:id="@+id/textView"
android:layout_width="0dp"
android:layout_height="66dp"
android:gravity="center"
android:text="TextView"
android:textSize="20sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.384"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.26" />

<Button
android:id="@+id/button2"
android:layout_width="133dp"
android:layout_height="70dp"
android:layout_marginBottom="396dp"
android:background="#F50057"
android:text="back"
android:textColor="@color/white"
android:textSize="24sp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.453"
app:layout_constraintStart_toStartOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```







Flags

Flags are defined in the Intent class that function as metadata for the intent. The flags may instruct the Android system how to launch an activity (for example, which task the activity should belong to) and how to treat it after it's launched (for example, whether it belongs in the list of recent activities).

IMPLICIT INTENTS:

Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

android:host — Specifies a valid hostname (e.g., google.com).

android:mimetype — Specifies the type of data your component is capable of handling.

For example, <type android:value=”vnd.android.cursor.dir/*”/> would match any Android cursor.

android:path — Specifies valid path values for the URI (e.g., /transport/boats/).

android:port — Specifies valid ports for the specified host.

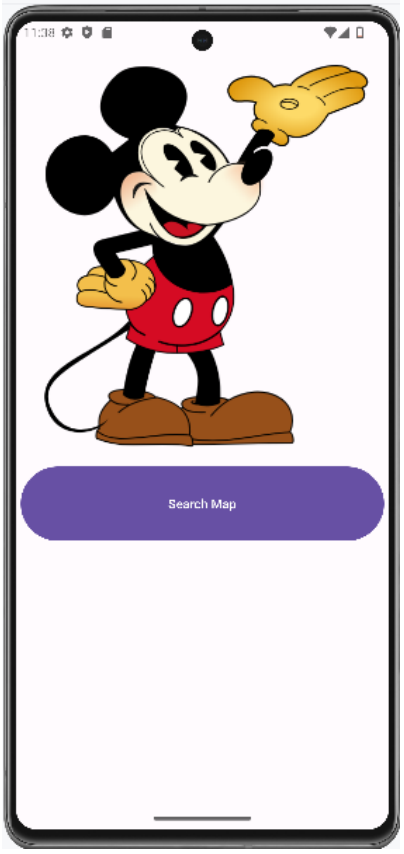
android:scheme — Requires a particular scheme (e.g., content or http).

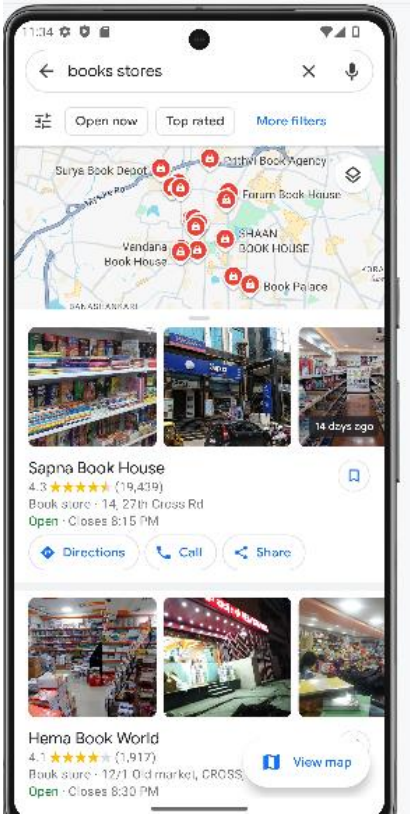
```
public class MainActivity extends AppCompatActivity {
Button b;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main2);
b=(Button) findViewById(R.id.button3);
b.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent i2=new Intent(Intent.ACTION_VIEW);
i2.setData(parse("https://www.youtube.com/watch?v=GQmq0OE_4ec"));//Y
outube
Intent i=new Intent(Intent.ACTION_PICK,
```

```
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI);//Gallery
        Intent mapIntent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("geo:12.9487,77.5724?q=books stores"));//Books stores Near NCB
        mapIntent.setPackage("com.google.android.apps.maps");
        startActivity(mapIntent);

          }
      });    }}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".MainActivity2">

<ImageView
android:id="@+id/imageView"
android:layout_width="match_parent"
android:layout_height="463dp"
app:srcCompat="@drawable/img" />



<Button
android:id="@+id/button3"
android:layout_width="401dp"
android:layout_height="90dp"
android:layout_gravity="center"
android:gravity="center"
android:text="Search Map" />

</LinearLayout>
```





EXPLICIT INTENTS:

Explicit intents specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.

Intent i = new Intent(getApplicationContext(), ActivityTwo.class);

Android Fragments

Android Fragment is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.

In Android, the fragment is the part of Activity which represents a portion of User Interface(UI) on the screen.

Following are important points about fragment −

- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- Fragments can't live on their own. They must be *hosted* by an activity or another fragment.
- 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-pane 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 behaviour that has no user interface component.

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.

The **FragmentManager** class is responsible to make interaction between fragment objects.

The Java class for a fragment needs to extend the Fragment base class:

```
public class Fragment1 extends Fragment {
}
```

Adding Fragments dynamically

To add fragments to an activity, you use the FragmentManager class by first obtaining an instance of it:

```
FragmentManagerfragmentManager = getFragmentManager();
```

You also need to use the FragmentTransaction class to perform fragment transactions (such as add, remove, or replace) in your activity:

FragmentTransactionfragmentTransaction = fragmentManager.beginTransaction();

In this example, the **WindowManager** is used to determine whether the device is currently in portrait mode or landscape mode.

Once that is determined, you can add the appropriate fragment to the activity by creating the fragment.

Next, you call the **replace()** method of the FragmentTransaction object to add the fragment to the specified view container.

In this case, android.R.id.content refers to the content view of the activity.

//---landscape mode---

Fragment1 fragment1 = new Fragment1();

// android.R.id.content refers to the content view of the activity

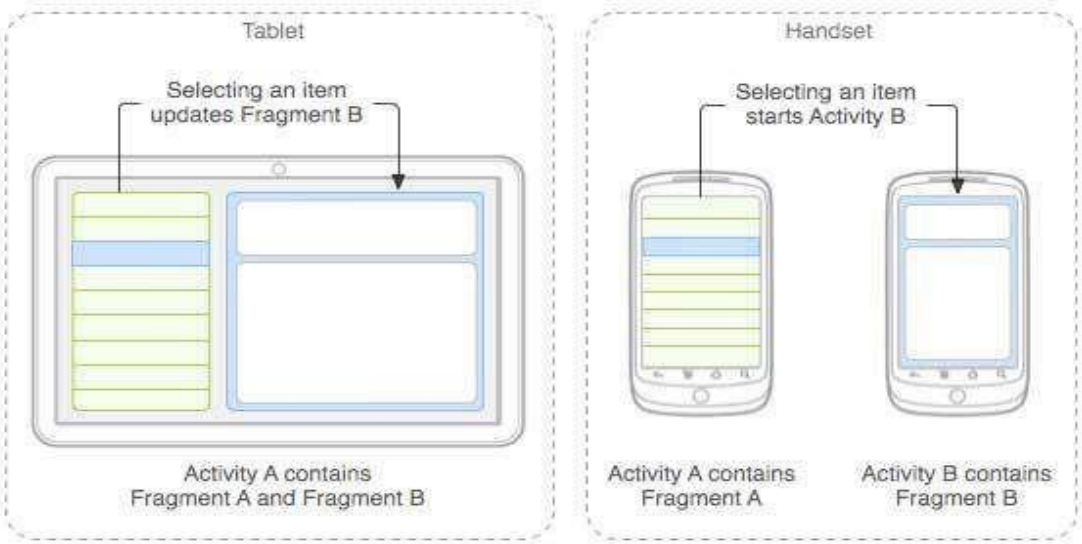
fragmentTransaction.replace(

android.R.id.content, fragment1);

Using the replace() method is essentially the same as calling the remove() method followed by the add() method of the FragmentTransaction object.

To ensure that the changes take effect, you need to call the commit() method:

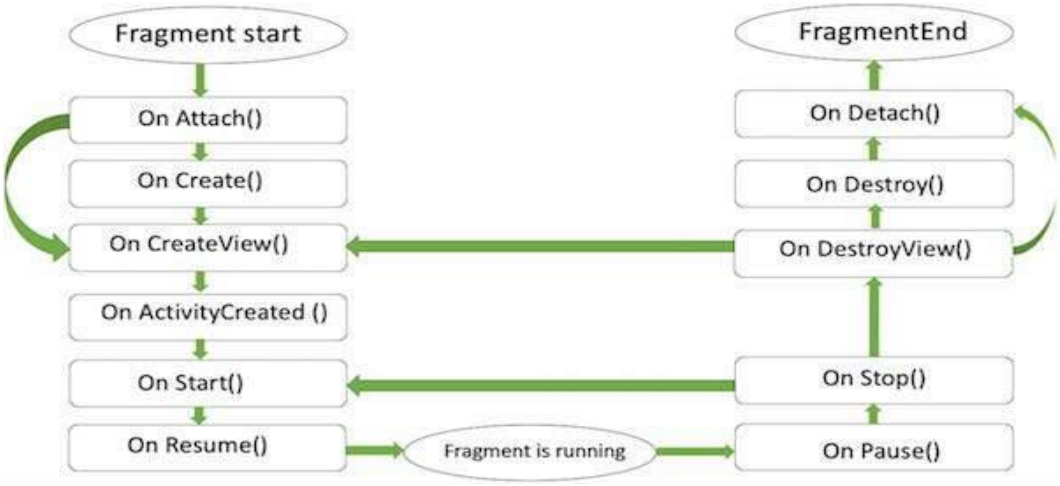
fragmentTransaction.commit();



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.

Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity.



Android Fragment Lifecycle Methods

No.	Method	Description
1)	onAttach(Activity)	it is called only once when it is attached with activity.
2)	onCreate(Bundle)	It is used to initialize the fragment.
3)	onCreateView(LayoutInflater, ViewGroup, Bundle)	creates and returns view hierarchy.
4)	onActivityCreated(Bundle)	It is invoked after the completion of

		onCreate() method.
5)	onViewStateRestored(Bundle)	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	onStart()	makes the fragment visible.
7)	onResume()	makes the fragment interactive.
8)	onPause()	is called when fragment is no longer interactive.
9)	onStop()	is called when fragment is no longer visible.
10)	onDestroyView()	allows the fragment to clean up resources.
11)	onDestroy()	allows the fragment to do final clean up of fragment state.
12)	onDetach()	It is called immediately prior to the fragment no longer being associated with its activity.

Create a fragment class:

To create a fragment, extend the class with Fragment class, and override its methods to insert your app logic, similar to the way you would create an Activity class. To create a minimal fragment that defines its own layout, provide your fragment's layout resource to the base constructor, as shown in the following example:

```
class ExampleFragment extends Fragment {
    public ExampleFragment() {
        super(R.layout.example_fragment);
    }
}
```

Add a fragment to an activity

Generally, your fragment must be embedded within an AndroidX FragmentActivity to contribute a portion of UI to that activity's layout. FragmentActivity is the base class for AppCompatActivity, so if you're already subclassing AppCompatActivity to provide backward compatibility in your app, then you do not need to change your activity base class.

You can add your fragment to the activity's view hierarchy either by defining the fragment in your activity's layout file or by defining a fragment container in your activity's layout file and then programmatically adding the fragment from within your activity. In either case, you need to add a FragmentContainerView that defines the location where the fragment should be placed within the activity's view hierarchy. It is strongly recommended to always use a FragmentContainerView as the container for fragments, as FragmentContainerView includes fixes specific to fragments that other view groups such as FrameLayout do not provide.

Add a fragment via XML:

To declaratively add a fragment to your activity layout's XML, use a FragmentContainerView element.

Here's an example activity layout containing a single FragmentContainerView:

```
<!-- res/layout/example_activity.xml -->
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.example.ExampleFragment" />
```

The android:name attribute specifies the class name of the Fragment to instantiate. When the activity's layout is inflated, the specified fragment is instantiated, onInflate() is called on the newly instantiated fragment, and a FragmentTransaction is created to add the fragment to the FragmentManager.

While your activity is running, you can make fragment transactions such as adding, removing, or replacing a fragment. In your FragmentActivity, you can get an instance of the FragmentManager, which can be used to create a FragmentTransaction. Then, you can instantiate your fragment within your activity's onCreate() method using FragmentTransaction.add(), passing in the ViewGroup ID of the container in your layout and the fragment class you want to add and then commit the transaction, as shown in the following example:

```
publicclassExampleActivityextendsAppCompatActivity{
   publicExampleActivity(){
      super(R.layout.example_activity);
   }
   @Override
   protectedvoid onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      if(savedInstanceState ==null){
         getSupportFragmentManager().beginTransaction()
            .setReorderingAllowed(true)
            .add(R.id.fragment_container_view,ExampleFragment.class,null)
            .commit();
      }
   }
}
```

The fragment transaction is only created when savedInstanceState is null. This is to ensure that the fragment is added only once, when the activity is first created. When a configuration change occurs and the activity is recreated, savedInstanceState is no longer null, and the fragment does not need to be added a second time, as the fragment is automatically restored from the savedInstanceState.

Fragment Manager:

Each Activity includes a Fragment Manager to manage the Fragments it contains.FragmentManager is the class responsible for performing actions on your app's fragments, such as adding, removing, or replacing them and adding them to the back stack. You can access the Fragment Manager using the getFragmentManager method:

```
FragmentManager fragmentManager = getFragmentManager();
```

The FragmentManager manages the fragment back stack. At runtime, the FragmentManager can perform back stack operations like adding or removing fragments in response to user interactions.
To display a fragment within a layout container, use the FragmentManager to create a FragmentTransaction. Within the transaction, you can then perform an add() or replace() operation on the container.
Ex:

```
FragmentManager fragmentManager =getSupportFragmentManager();
fragmentManager.beginTransaction()
   .replace(R.id.fragment_container,ExampleFragment.class,null)
```

```
.setReorderingAllowed(true)
.addToBackStack("name")// Name can be null
.commit();
```

Fragment Transactions:

Fragment Transactions can be used to add, remove, and replace Fragments within an Activity at runtime. Each set of changes is committed together as a single unit called a FragmentTransaction.

Using Fragment Transactions, you can make your layouts dynamic — that is, they will adapt and change based on user interactions and application state. A new Fragment Transaction is created using the begin Transaction method from the Activity's Fragment Manager.
Modify the layout using the add, remove, and replace methods, as required, before setting the animations to display. When you are ready to execute the change, call commit to add the transaction to the UI queue.

```
FragmentTransaction  fragmentTransaction=fragmentManager.beginTransaction();
// Add, remove, and/or replace Fragments.
// Specify animations.
fragmentTransaction.commit ();
```

Adding,Removing and Replacing Fragments:

When adding a new UI Fragment, specify the Fragment instance to add, along with the container View into which the Fragment will be placed.

```
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
fragmentTransaction.add(R.id.ui_container, new MyListFragment());
fragmentTransaction.commit();
```

To remove a Fragment, you first need to find a reference to it, using the FragmentManager's findFragmentById method. Then pass the found Fragment instance as a parameter to the remove method of a Fragment Transaction:

```
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
Fragment fragment = fragmentManager.findFragmentById(R.id.details_fragment);
fragmentTransaction.remove(fragment);
```

```
fragmentTransaction.commit();
```

You can also replace one Fragment with another. Using the replace method, specify the container ID containing the Fragment to be replaced, the Fragment with which to replace it, and (optionally) a tag to identify the newly inserted Fragment.

```
FragmentTransaction fragmentTransaction =fragmentManager.beginTransaction();
fragmentTransaction.replace(R.id.details_fragment,newDetailFragment(selected_index));
fragmentTransaction.commit();
```

Example of Fragments:

Main_activity.java:

```
public class MainActivity extends AppCompatActivity {
FrameLayout frameLayout;
TabLayout tabLayout;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
frameLayout = (FrameLayout) findViewById(R.id.framelayout);
tabLayout = (TabLayout) findViewById(R.id.tablayout);
FragmentManager fragmentManager = getSupportFragmentManager();
fragmentManager.beginTransaction()
                .replace(R.id.framelayout, new Fragment_First(),null)
                .setReorderingAllowed(true)
                .addToBackStack("name") // Name can be null
.commit();
tabLayout.setOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
@Override
public void onTabSelected(TabLayout.Tab tab) {
Fragment f=null;
switch(tab.getPosition())
                {
case 0: f=new Fragment_First();
break;
case 1: f=new Fragment_Second();
break;
case 2:f=new Fragment_Third();
                }

fragmentManager.beginTransaction()
```

```
                    .replace(R.id.framelayout, f,null)
                    .setReorderingAllowed(true)
                    .addToBackStack("name") // Name can be null
.commit();
            }

@Override
public void onTabUnselected(TabLayout.Tab tab) {

            }

@Override
public void onTabReselected(TabLayout.Tab tab) {

            }
        });
    }
}
public class Fragment_First extends Fragment {

// TODO: Rename parameter arguments, choose names that match
// the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
private static final String ARG_PARAM1 = "param1";
private static final String ARG_PARAM2 = "param2";

// TODO: Rename and change types of parameters
private String mParam1;
private String mParam2;

public Fragment_First() {
// Required empty public constructor
}

/**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment Fragment_First.
     */
    // TODO: Rename and change types and number of parameters
public static Fragment_First newInstance(String param1, String param2) {
Fragment_First fragment = new Fragment_First();
Bundle args = new Bundle();
args.putString(ARG_PARAM1, param1);
args.putString(ARG_PARAM2, param2);
```

```
fragment.setArguments(args);
return fragment;
    }

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
if (getArguments() != null) {
mParam1 = getArguments().getString(ARG_PARAM1);
mParam2 = getArguments().getString(ARG_PARAM2);
        }
    }

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment__first, container, false);
    }
}
```

Create similar two fragments Fragment_Second.java and Fragment_Third.Java

activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:paddingVertical="30dp"
tools:context=".MainActivity">

<com.google.android.material.tabs.TabLayout
android:id="@+id/tablayout"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:background="#389AC6"
app:tabGravity="fill"
app:tabIndicator="@color/white"

app:tabSelectedTextColor="#FF0000"
app:tabTextColor="@color/white">

<com.google.android.material.tabs.TabItem
```

```
android:id="@+id/tab1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="First">

</com.google.android.material.tabs.TabItem>

<com.google.android.material.tabs.TabItem
android:id="@+id/tab2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Second">

</com.google.android.material.tabs.TabItem>

<com.google.android.material.tabs.TabItem
android:id="@+id/tab3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Third" />
</com.google.android.material.tabs.TabLayout>

<FrameLayout
android:id="@+id/framelayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_marginTop="30dp"
android:fadingEdge="horizontal|vertical"
android:isScrollContainer="true"
android:padding="25dp" />

</LinearLayout>
```

Fragment_first.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".Fragment_Second">

<TextView
android:id="@+id/t1"
android:layout_width="match_parent"
android:layout_height="213dp"
android:gravity="center"
```

```
android:text="The National College Basavanagudi"
android:textAppearance="@style/TextAppearance.AppCompat.Body1"
android:textSize="48sp"
android:textStyle="bold" />

<ImageView
android:id="@+id/imageView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:adjustViewBounds="true"
android:src="@drawable/img" />
</FrameLayout>
```

Design fragment_second and fragment_third.xml similiat to fragment_first and change the text names in both xml files.