1.What is Android Studio and Explain its features?

Introduction to Android OS

What is Android?

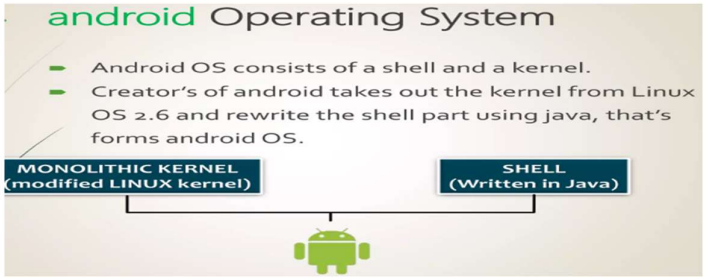
Android is an **open source** and **Linux-based Operating System** for mobile devices such as smartphones and tablet computers.

Android is a stripped version of Linux Operating System i.e. it is compiled to run on **smart phones and tablets which** have **fewer resources** in terms of **processing power and memory than a personal computer**.

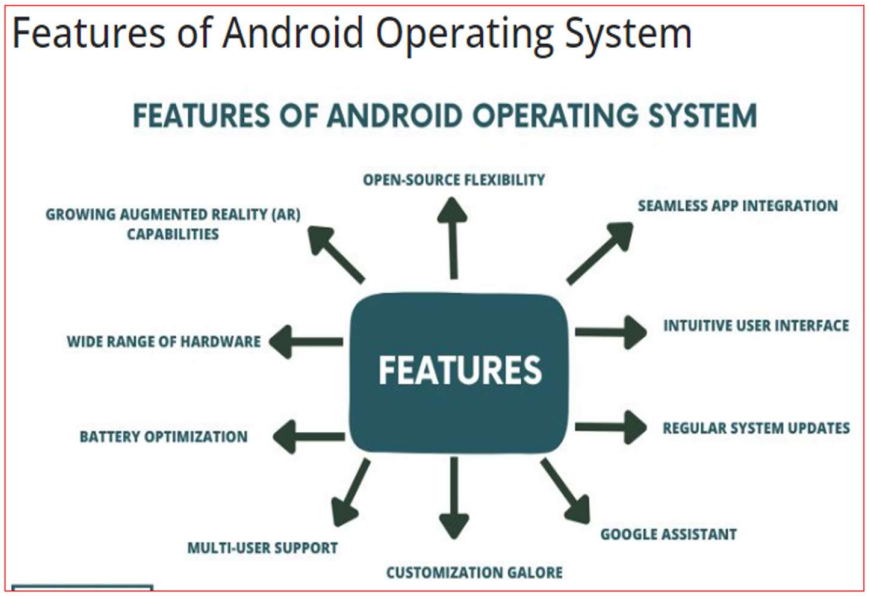
However, it is still capable of doing daily productive jobs like word processing, sending emails , syncing , sharing , calling and other needs of communication. It had also become a main device of entertainment.

We can **stream videos, listen to music and play so many games** by downloading apps from Google Play Market. **Many of them are free**.

This operating system is Open Source and many **phone manufacturers like Samsung, HTC, LG have launched new android devices both phones and tablets based on Android**.



Features of Android OS



1. Open-Source Flexibility

At the core of Android's appeal lies its open-source nature. Unlike its counterparts, Android welcomes collaboration and modification by developers, leading to a dynamic ecosystem that constantly evolves. This flexibility empowers **developers to create customized solutions and users to personalize their devices** to match their preferences.

2. Seamless App Integration

One of Android's standout features is its seamless integration with a vast array of applications. With the **Google Play Store serving as a treasure trove** of options, users can choose from an extensive selection of apps to enhance their experience. From productivity tools to entertainment platforms, Android ensures that users can tailor their devices to suit their lifestyles seamlessly.

3. Intuitive User Interface

Android's user interface (UI) has undergone significant refinements over the years, culminating in an intuitive and **user-friendly experience**. The UI design prioritizes accessibility, ensuring that even novice users can navigate the system with ease. With a focus on simplicity and clarity, Android's UI sets a benchmark for usability across different age groups and demographics.

4. Regular System Updates

The Android operating system is dedicated to **continuous improvement through regular system updates**. These updates not only introduce new features but also **enhance security protocols** and address any potential vulnerabilities. This commitment to keeping devices up-to-date ensures a secure and optimized experience for all users.

5. Google Assistant – Your AI Companion

Among the standout features is the integration of **Google Assistant, an AI-powered virtual companion** designed to streamline tasks and provide real-time information. With **voice commands, users can set reminders, send messages, play music, and even control smart home devices.** The ever-evolving capabilities of Google Assistant make it a valuable addition to the Android ecosystem.

6. Customization Galore

Android grants users the **freedom to customize their devices** extensively. From changing the launcher and theme to installing third-party widgets, the level of personalization is unparalleled. This feature not only reflects individuality but also enhances productivity by tailoring the device's functionality to specific needs.

7. Multi-User Support

Android recognizes the diverse needs of its users, offering multi-user support on devices such as tablets. This feature is particularly **beneficial for families**, as **each member can have a personalized profile, complete with apps and settings**. It ensures a shared device can cater to different preferences without compromising on privacy.

8. Battery Optimization

Efficient battery management is a hallmark of the Android OS. With features like **Adaptive Battery**, the system learns usage patterns to **optimize power allocation to different apps**. This translates to **extended battery life**, ensuring that users can stay connected and engaged throughout their day without worrying about running out of power.

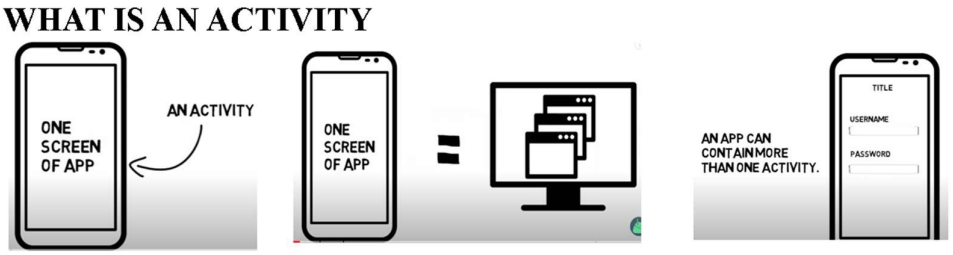
9. Wide Range of Hardware

Android's compatibility with a wide range of hardware is a significant advantage. Whether it's a budget-friendly device or a flagship model, the **Android OS seamlessly adapts to varying specifications**. This inclusivity enables users to choose devices that align with their requirements and budget without sacrificing the operating system's quality.

10. Growing Augmented Reality (AR) Capabilities

As technology advances, Android remains at the forefront of **integrating augmented reality into the user experience**. With ARCore, Google's platform for building augmented reality experiences, Android devices can provide immersive and interactive encounters that **bridge the gap between the digital and physical worlds**.

2.Define Activity and Explain Activity Lifecycle?



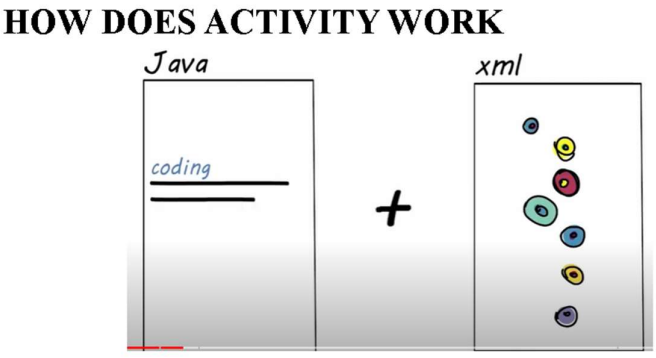
- An activity is **one screen of an app**.
- In that way the activity is **very similar to a window** in the Windows operating system or like **a web page of a website**.
- The most **specific building block of the user interface** is the activity.
- An Android app contains activities, meaning **one or more screens**.

  Examples: **Login screen, sign up screen, and home screen**.

Activity is nothing but **a java class in Android which has some pre-defined functions** which are triggered at different App states

Every application which has UI must inherit it to create a window.

Whenever we open any application, one of its activity opens, its called the Launcher Activity.
When you create a new project, android studio creates an activity by name "**MainActivity.java**" and its XML file my name "**activity_main.xml**". It is the activity that opens up when an app is opened.



- An activity in Android is a specific **combination of XML files and JAVA files**.
- It is basically a **container that contains the design as well as coding stuff.**
- **XML files provide the design of the screen and JAVA files deal with all coding stuff** like handles, what is happening, design files, etc
- JAVA files and XML files make your **activity complete.**

Activity Life Cycle / LIFECYCLE OF ACTIVITY

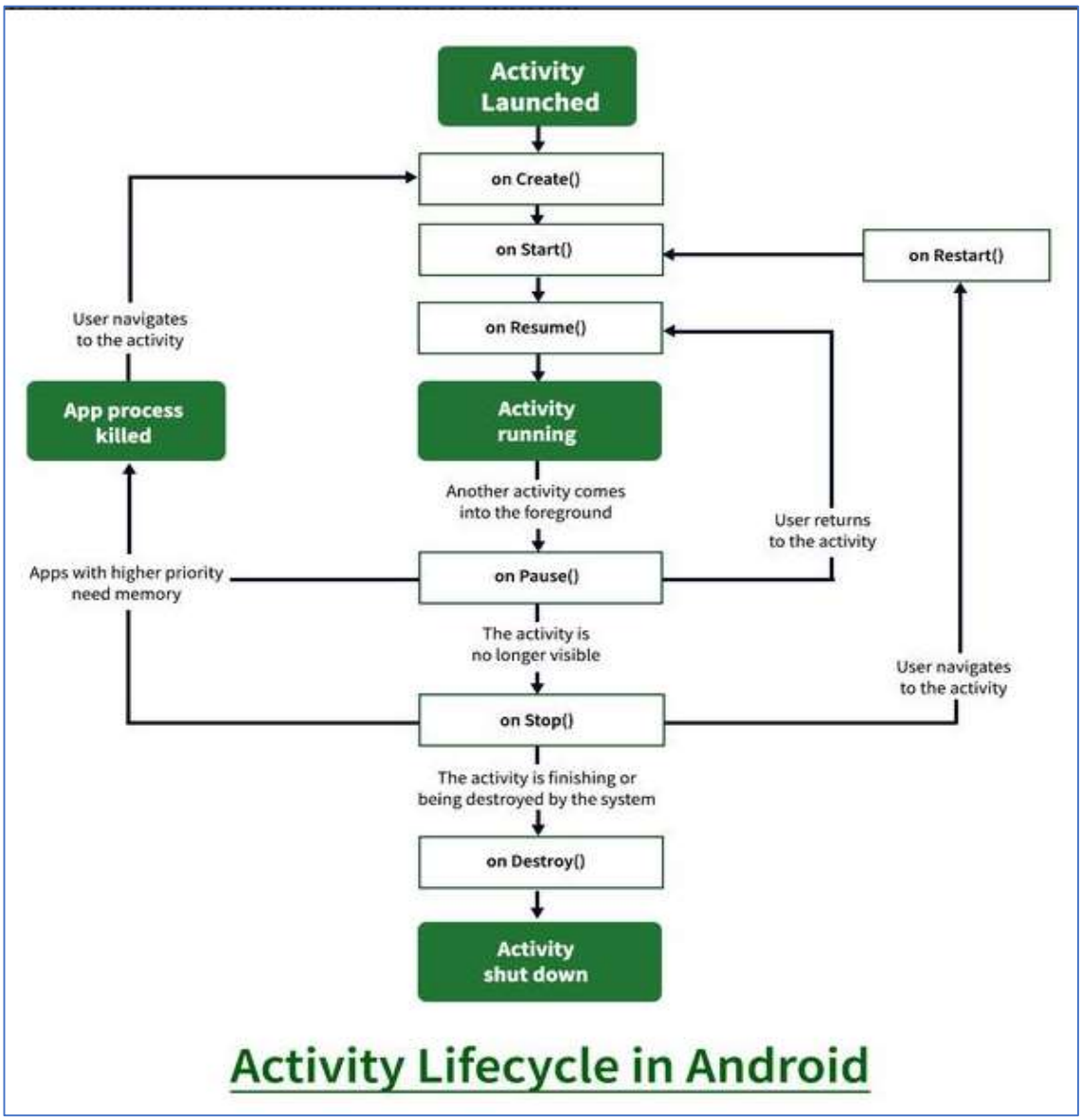
As a user **navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle**.

The **Activity class provides several callbacks that allow the activity to know that a state has changed**, that the system is **creating, stopping, resuming, or destroying** an activity. Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity.

For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user minimizes the app. When the user returns, you can reconnect to the network and allow the user to resume the video from the same spot.

Each callback allows you to perform specific work that is well suited for a given change of state.

Doing the right work at the right time and handling transitions properly make your app more robust and performant and helps make the user experience better.



1. onCreate( )

This callback is fired when the system **first creates the activity**. In onCreate method, all such operations are performed which should be done **only once for the entire life of the activity**.

This method has a parameter ***savedInstanceState*** **which is a bundle object**. It contains activity's previous saved state. If the activity has never existed before, the value of Bundle object is null.

```
public class MainActivity extends AppCompatActivity {

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

2. onStart( )

When the activity enters the Started state, the system invokes this callback. The *onStart( )* call makes the **activity visible to the user**, as the **app prepares for the activity to enter the foreground** and become interactive.

This method can be used **to initialize the code that maintains the user interface**. After *onStop( )* , if activity **restarts then *onStart( )*** is called.

***onCreate ( )* function is called only once but onStart( ) can be called multiple times** , when activity enters the started state.

3. onResume( )

The ***onStart( )* method completes very quickly** and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the **activity enters the *Resumed* state**, and the system invokes the *onResume( )* method.

This is the state in which the **app interacts with the user**. The app stays in this state until **something happens to take focus away from the app**. Such an event might be, for instance,

receiving a phone call,
the user's navigating to another activity, or
the device screen's turning off.

When such an event occurs, the activity **enters the *Paused* state**, and the system invokes the *onPause( )* callback.

4. onPause( )

The system calls this method as the first indication that the **user is leaving the activity** (though it does not always mean the activity is being destroyed); it indicates that **the activity is no longer in the foreground** (though it may still be visible if the user is in multi-window mode).

In most case onPause() method called by Android OS when user press **Home button (Center Button on Device) to make hide**. Activity is **not visible to user and goes in background** when onPause() method is executed

When the activity moves to the paused state, any lifecycle-aware **component tied to the activity's lifecycle will receive the *onPause( )* event**. This is where the **lifecycle components can stop any functionality that does not need to run while the component is not in the foreground**

***onPause( )* method can be used** to **release system resources**, handles to sensors (like GPS), or any resources that **affect battery life** while your activity is paused and the user does not need them.

5. onStop( )

When your activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the onStop( ) callback. Any activity gets stopped in case some other activity takes place of it. For example, if a user was on screen 1 and click on some button and moves to screen 2. In this case Activity displaying content for screen 1 will be stopped.

The system may also call onStop( ) when the activity **has finished running, and is about to be terminated**. In the *onStop( )* method, the app should **release or adjust resources that are not needed** while the app is not visible to the user. It should be used to perform relatively **CPU-intensive shutdown operations.** So Activity will be in stopped state when hidden or replaced by other activities that have been launched or switched by user

6. onRestart( )

onStart( ) will always be called whenever you enter your Activity just after *onCreate( )* but the *onRestart( )* will only be called before *onStart( )* when your **Activity comes from being stopped** (passing from *onStop( )*) back to the vision.

7. onDestroy( )

onDestroy( ) is called before the activity is destroyed. The system invokes this callback either because:

1. if user **pressed the back navigation button** then activity will be destroyed after **completing the lifecycle of pause and stop**.
2.In case if user press the home button and app moves to background. User is not using it no more and it's being shown in recent apps list. So in this case **if system required resources need to use somewhere else then OS can destroy the Activity.**

After the Activity is destroyed if user again click the app icon, in this case activity will be recreated and follow the same lifecycle again.

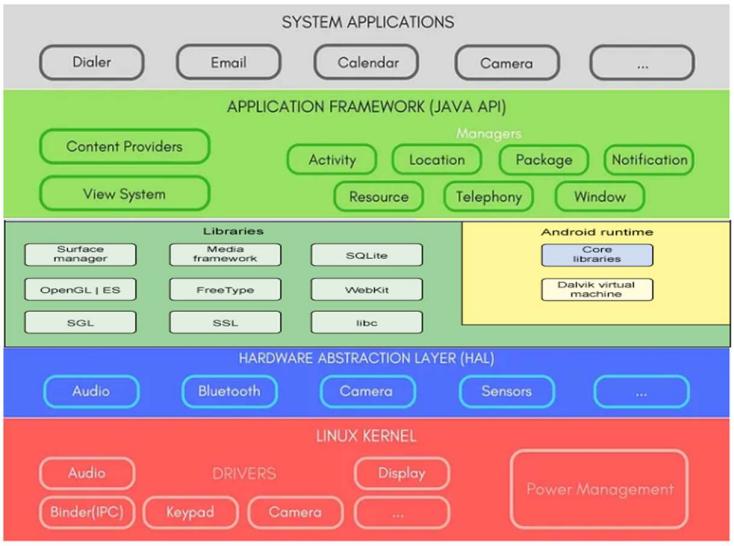
When you open the app it will go through below states:
onCreate() –> onStart() –> onResume()
When you press the back button and exit the app
onPaused() — > onStop() –> onDestory()
When you press the home button
onPaused() –> onStop()
After pressing the home button, again when you open the app from a recent task list
onRestart() –> onStart() –> onResume()
After dismissing the dialog or back button from the dialog
onResume()
If a phone is ringing and user is using the app
onPause() –> onResume()
After the call ends
onResume()
When your phone screen is off
onPaused() –> onStop()
When your phone screen is turned back on
onRestart() –> onStart() –> onResume()

3.Explain Android Development Framework with block diagram.

Android Architecture

The Android framework is a set of software components that provide the foundation for building Android applications.

It is divided into **five layers**:



4. Android Framework

This layer provides **the basic building blocks for Android applications** The Android Framework is a **collection of Java classes and libraries** that provide the core functionalities and building blocks for Android applications. It includes classes for managing the user interface, accessing hardware resources, and communicating with other applications

It consists of various modules:

- **Activity Manager**: Manages the lifecycle of Android applications and their components, such as activities and services.
- **Content Providers**: Allow apps to share data and access it securely. They act as a bridge between applications and databases.
- **Notification Manager**: Handles notifications generated by apps and system events.
- **Location Manager**: Enables apps to access location-based services.
- **Connectivity Manager**: Manages network connections, including Wi-Fi and mobile data.

The Android Framework also includes the **Application Framework**, which allows developers to **build custom applications using pre-built components.**

Android development framework

In general, **the basic method of creating an Android app is to use Android Studio** (As it is the **official IDE for Android**).

However, there are **many other Android Development Frameworks** that are quite popular.

There are several types of mobile application development frameworks, each **catering to different needs and platforms**. Here are some common categories

1.Native Development Frameworks: These frameworks are platform-specific, allowing developers to build apps for a single platform using the platform's native language and tools. Examples include:
iOS Development (Swift/Objective-C): For Apple devices, Swift and Objective-C are used for native iOS app development.
Android Development (Java/Kotlin): For Android, Java and Kotlin are the primary languages for building native apps.

2. Cross-Platform Development Frameworks: These frameworks enable developers to create apps for multiple platforms using a single codebase. Examples include:

React Native: Developed by Facebook, React Native allows developers to build mobile apps for iOS and Android using JavaScript and React. It's used by companies like **Facebook, Instagram, and Airbnb.**
Flutter: Created by Google, Flutter is another cross-platform framework using Dart, and it's known for building beautiful, natively compiled applications. Apps like **Alibaba** and **Google Ads** have been developed using Flutter.

3. Hybrid Mobile App Development Frameworks: Hybrid mobile app development frameworks combine native and web technologies, usually relying on web views to render content within a native app. Examples include:
Apache Cordova (PhoneGap): It allows developers to use HTML, CSS, and JavaScript to build cross-platform apps. **Adobe PhoneGap** is a popular variation.
Ionic: Built on top of Cordova, Ionic provides a UI framework and tools for building hybrid apps.

4. Game Development Frameworks: These frameworks are specialized for creating mobile games. Examples include:
Unity: It's a versatile game engine that supports mobile platforms like iOS and Android. Games like **"Pokémon GO"** and **"Super Mario Run"** were built using Unity.

5.Detail what are various types of android applications?

Types of Android application

1. Native Apps:

Native Apps are **written in a specific programming language to work on a particular Operating system**.

A majority of the smartphones either run an Android OS or the iOS if it's an apple device. Native Apps are **specifically built for specific OS** to make the most of the functionalities of the devices that run the particular OS. And hence native apps cannot be used on **different types of operating system**. Blackberrry, Symbian, and Windows phone are functional only on their respective platform.

Technology Used: **Xcode and Objective-C** are mainly used for **iOS apps**, and **Eclipse, Kotlin and Java** are used to build **Android apps.**

Native apps are generally built to make the **most of all the features and tools of the phones such as contacts, camera, sensors**, etc. Native apps ensure a high performance and elegant user experience as the developers use the **native device UI to build apps.**

Native apps are easily available on the OS specific apps stores.For example you will find **native Android apps on Google Play Store**, **iOS apps on App Store**, and **windows apps on the microsoft store**

Advantages of native apps:

1. Fast performance due to simple code specific to device and OS.
2. Better use of OS and device specific functionalities.
3. Interactive UI/UX.
4. Lesser compatibility issues and faster to configure.
5. Offline connection

Disadvantages of native apps:

1. Building OS specific apps can be time-consuming
2. OS specific programming languages like swift and java are hard to learn.
3. Longer release cycles to ensure stability.
4. Requires separate codebase to add new features.

Few Popular native mobile applications

2. Web Apps

Web apps behave similarly to native apps but are **accessed via a web browser** on your mobile device. They're **not standalone apps** in the sense of having to **download and install code** into your device.

It **does not require any storage space or installation process to use the app**. Mobile web **apps adapt to various screen sizes and devices easily**.

One of the major differences between the native apps and mobile apps, is that native mobile apps can function both in the **offline mode and the online mode**, whereas the web apps require an active internet connection for them to work. Since these apps are not installed on your smartphone, there is **no need for updating the web app as they update themselves on the web-hosted servers.**

Technology Used: Web apps are designed using **HTML5, CSS, JavaScript, React, Ruby**, and similar programming languages used for web work.

Advantages of web apps:

1. Reduced business cost.
2. No installation needed.
3. Better reach as it can be accessed from anywhere.
4. Always up-to-date.

Few Mobile Web Apps

Disadvantages of web apps:

1. Web apps fail to work when you are offline.
2. Limited number of functionalities as compared to Native apps.
3. It takes a longer time to develop.
4. Security risk.

3. Progressive Web Apps (PWAs)

Progressive Web Apps (PWAs) **are extensions of the website** that you can save on your computer systems or devices and use like an app. When you come across the **option to "install" a web app, it often simply bookmarks the website URL on your device.**

One kind of web app is the **progressive web app (PWA), which is basically a native app running inside a browser**. PWAs use web browser APIs and functionalities to bring a **native app-like experience across devices**. It is a type of a webpage that can be added onto your devices or computer systems to mimic a web application. The PWAs run fast regardless of the Operating Systems and devices types.

Advantages of Progressive web apps:

1. They use very little data – An app which takes close to 10 MBs as a native app, can be reduced to about 500KB when made a PWA.
2. PWAs get updated like web-pages. They automatically get updated every time you use them.
3. There is no need for installation as PWAs are simple web-pages. Users choose to 'install' when they like it.
4. You can easily share PWAs by simply sending its URL.

Disadvantages of progressive apps:

1. There are limitations to using all the Hardware and Operating Systems features.
2. PWAs can pose a few hardware integration problems.
3. Full support is not available in default browsers of some of the manufacturer's.

4. Hybrid Apps

Hybrid apps combine **the best of both native and web apps**. The hybrid apps are **written using HTML, Javascript, and CSS web technologies and work across devices running different OSs**.

Development teams will not need to struggle with Objective-C or Swift to build native apps anymore, and use standard web technologies like Javascript, Angular, HTML and CSS.

The **mobile development framework Cordova wraps the Javascript/HTML code and links the hardware and functions of the device.**

Hybrid Apps are **built on a single platform and distributed across various app stores such as Google Play store or Apple's app store similar to Native apps**.

Hybrid apps are best used when you want to build apps that **do not require high-performance, full device access apps**.

Advantages of hybrid apps:

1. Easy to build
2. Shareable code makes it cheaper than a native app
3. Easy to push new features since it uses a single code base.
4. Can work offline.
5. Shorter time to market, as the app can be deployed for multiple OSs.

Disadvantages of Hybrid apps:

1. Complex apps with many functions will slow down the app.
2. More expensive than web apps
3. Less interactive than native apps
4. Apps cannot perform OS specific tasks

Examples of Hybrid Apps:

6.Discuss the importance of Android manifest file and what are the essential tags available in manifest file.

Android Manifest file

The Android Manifest is **an XML file which contains important metadata about the Android app.**

Every android project that we make, **need to have an Android Manifest file within**. This file is present in the **root of the project source set**.

It describes all the **important information of the application to the Android build tools**.

This includes the **package name, activity names, main activity (the entry point to the app), Android version support, hardware features support, permissions, and other configurations**

The **information that is stored in the Manifest file** is as follows:

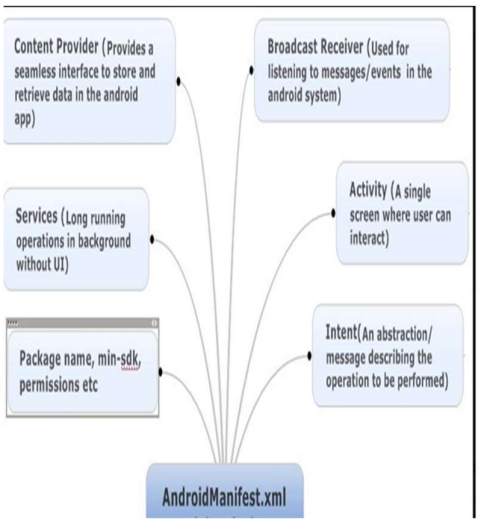
- The **name of the application's package**, it is generally the **code's namespace**. This information is used to determine the location of the code while building the project.
- Another component is the one, that includes all the **activities, services, receivers, and content providers**.
- The **permissions that are required by the application** to access the protected parts of the system and other apps.
- The features required by the app, that affect **which devices can install the app from Google Play**. These features include both hardware and software features.
- It also specifies the application metadata, which includes the **icon, version number, themes, etc**.

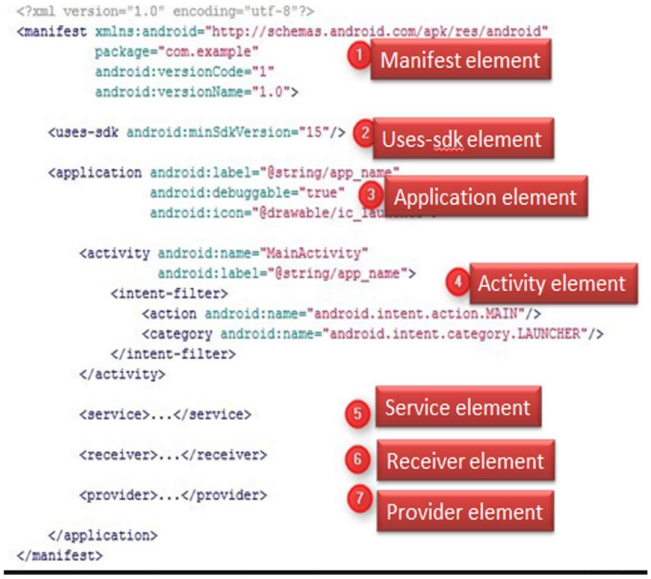
- <?xml version="1.0" encoding="utf-8"?>
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <application
      android:allowBackup="true"
      android:dataExtractionRules="@xml/data_extraction_rules"
      android:fullBackupContent="@xml/backup_rules"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/Theme.Week3A"
      tools:targetApi="31">
      <activity
        android:name=".MainActivity"
        android:exported="true">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />

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

  </manifest>





Element Tags of Manifest File

Following are the essential element tags of Manifest.xml files:

1. <manifest>

It is the **root element of this element**. It consists of a package attribute package that tells the activity's package name.

2. <application>

It is the **sub-element of the manifest file** that includes the declaration of the namespace. It contains certain attributes. These attributes declare the application components, and these attributes include:

- **android:icon** represents the icon for all the android application components.
- **allowBackup**
- **android:label** works as the default label for all the application components.
- **android:theme** represents a common theme for all the android activities.

3. <activity>

Activity is a **sub-element of application**. It has the declaration of the activity that must be there in the manifest file. It also has certain attributes like name, label, theme, etc.

- **android:label** represents a label i.e. displayed on the screen.
- **android:name** represents a name for the activity class. It is required attribute.

4. <intent-filter>

It is an **element in the activity**, it describes the type of intent in which the Android components can respond.

5. <action>

This element provides an **action for the intent filter**. Each intent filter **must have at least one action element** in it.

6. <category>

This element **adds the category name** in an intent-filter.

7. <service>

This element contains the operations that are provided by libraries or APIs.

Manifest File Application Property Elements

A list of important Application Property Elements in manifest.xml (Sub- Node Elements)

1. **<uses-permission>:** This element specifies the Android Manifest permissions that are requested for the purpose of security.
2. **<permission>:** This element sets permission to provide access to control for some components of the app.
3. **<permission-groups>:** This element sets permission to provide access to control for a set of components of the app.
4. **<permission-tree>:** This element refers to a specific component that is the owner of the set of components.
5. **<instrumentation>:** This element tells the interaction between the app and the system.

6. **<uses-sdk>:** This one specifies the compatibility of the app.
7. **<uses-configuration>:** This element specifies the permissions that are requested for the purpose of security.
8. **<uses-feature>:** This element specifies one hardware or software feature that is required by the Application.
9. **<supports-screen, compatible-screen>:** These elements tell the screen size and configuration.

7.Define layout Explain Linear Layout with example.

Layouts

Layout basically refers to the **arrangement of elements on a page** these elements are likely to be images, texts or styles. These are a part of **Android Jetpack**. They define the structure of android user interface in the app, like in an activity. **All elements in the layout are built with the help of Views and ViewGroups**. These layouts can have various widgets (components) like buttons, labels, textboxes, and many others. **Layouts are container views (and, therefore, subclassed from ViewGroup) designed to control how child views are positioned on the screen**.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/layout2“
        android:layout_width="match_parent“
        android:layout_height="match_parent"
        android:layout_weight="1“
        android:background="#8ED3EB“
        android:gravity="center“
        android:orientation="vertical">

            <TextView android:id="@+id/textView4“
                android:layout_width="match_parent“
                android:layout_height="wrap_content“
                android:layout_marginLeft="10dp“
                android:layout_marginTop="-40dp“
                android:fontFamily="@font/almendra_bold"
                android:text="This is a TextView" />
</LinearLayout>
```

1. Linear Layout

We use this layout to **place the elements in a linear manner**. A Linear manner means **one element per line**. This layout creates various kinds of forms on Android. In this, arrangement of the elements is in a **top to bottom manner**.

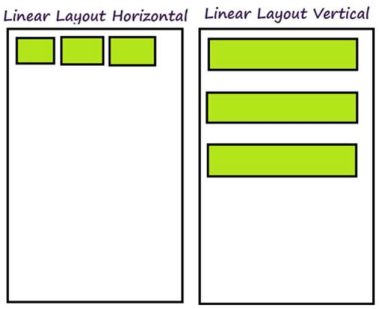
Linear layout is simple and easy to use, it creates a scroll bar if the length of the window exceeds the length of the screen.

Vertically linear layout has only one item per row. **Linear layout has many different attributes** which can be used to customize linear layout according to needs.

This can have two orientations:

a. Vertical Orientation – It is shown above where the widgets such as TextViews, and all in a vertical manner.

b. Horizontal Orientation – It is shown above where the widgets such as TextViews, and all in a horizontal manner.



Attributes of Linear Layout

android:layout_weight	It is defined individually to the child's views to specify how Linear Layout **divides the remaining space amongst the views it contains**
android:weightSum	Defines the total weight sum
android:orientation	How the elements should be arranged in the layout. It can be horizontal or vertical.
android:gravity	It specifies how an object should position its content on its X and Y axes. Possible values are – center_vertical, fill, center, bottom, end, etc.
android:layout_gravity	Sets the gravity of the View or Layout relative to its parent. Possible values are – center_vertical, fill, center, bottom, end, etc.
android:baselineAligned	This must be a Boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
android:id	This gives a unique id to the layout.

Arrange children views in a vertical manner

```
<?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:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity">
   <!-- Add vertical in the android:orientation-->
   <!-- Add Button-->
   <Button
      android:layout_width="match_parent"
      android:layout_margin="10dp"
      android:layout_height="wrap_content"/>
   <!-- Add Button-->
   <Button
      android:layout_width="match_parent"
      android:layout_margin="10dp"
      android:layout_height="wrap_content"/>
   <!-- Add Button-->
   <Button
      android:layout_width="match_parent"
      android:layout_margin="10dp"
      android:layout_height="wrap_content"/>
 </LinearLayout>
```

Arrange children views in a horizontal manner

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   tools:context=".MainActivity">

   <!-- Add horizontal in the android:orientation-->
   <!-- Add Button-->
          <Button
             android:layout_width="wrap_content"
             android:layout_height="match_parent"
             android:layout_margin="10dp" />
          <!-- Add Button-->
          <Button
             android:layout_width="wrap_content"
             android:layout_height="match_parent"
             android:layout_margin="10dp" />
          <!-- Add Button-->
          <Button
             android:layout_width="wrap_content"
             android:layout_height="match_parent"
             android:layout_margin="10dp" />
</LinearLayout>
```

8.What is the use of externalization resources in android.Discuss on color.xml and string.xml in detail with example for each.

External Resources in Android

- It's always good practice to keep **non-code resources like images and string constants** external to your code.
- Android supports the **externalization of resources ranging from simple values such as strings and colors to more complex resources like images (Drawables), animations, and themes.**
- These resources are always maintained separately in various sub-directories under **res/ directory** of the project.
- You should also provide alternative resources for specific device configurations, by grouping them in specially-named resource directories.
- At runtime, Android uses the appropriate resource based on the current configuration. For example, you might want to provide a different UI layout depending on the screen size or different strings depending on the language setting.

Organize resource in Android Studio

```
MyProject/
  app/
    manifest/
      AndroidManifest.xml
    java/
      MyActivity.java
    res/
      drawable/
        icon.png
      layout/
        activity_main.xml
        info.xml
      values/
        strings.xml
```

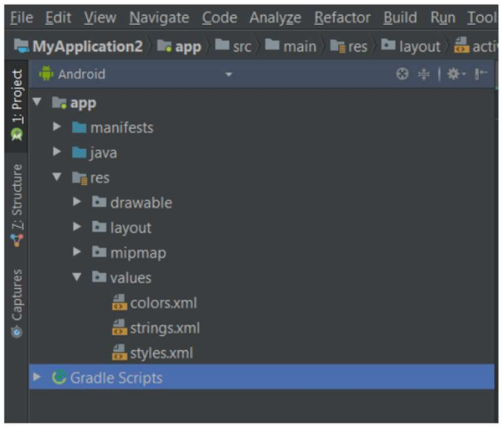
Simple Values – colors.xml, strings.xml

String.xml file contains all the strings which will be **used frequently in Android project**.

String.xml file present in the values folder which is **sub folder of res folder** in project structure.

In Android Studio, we have many Views such as TextView,Button,EditText,CheckBox,RadioButton etc.

These views have a common attribute named text. **we can assign value to this attribute directly or via string.xml.**



In large applications, we use **some text many times in our projec**t so rather than define that particular text many times. it **is best to store in string.xml file and call it by its string name in project.**

For example: Suppose we have a text:- "Hey How are You" and print them many times

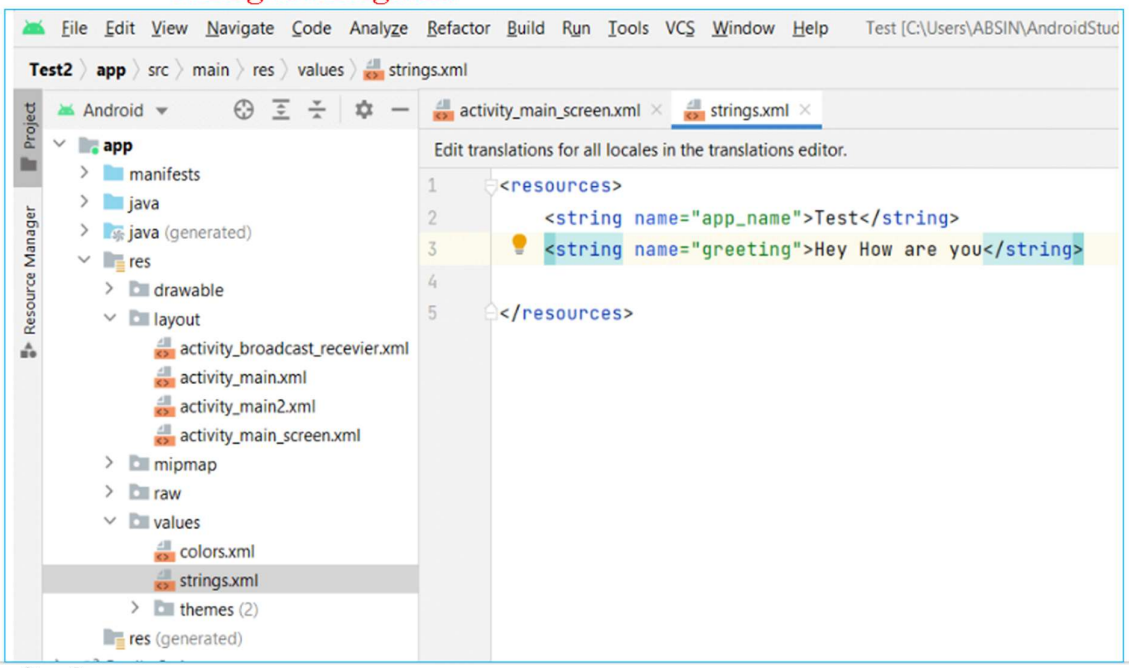
Best way is to store this text in string.xml once and call string wherever required. This will save a lot of efforts and less line of code.

Step 1

Go to String.xml file (app->res->values->strings.xml)

Step 2

Now define a **String in string.xml.**



Step 3

Now Go to layout XML file and added this code .Here we **call this string by using *@string/greeting*.**

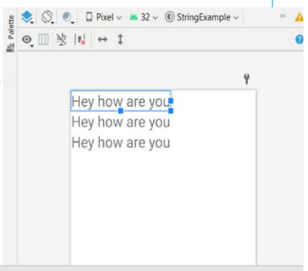
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/greeting"
        android:textSize="30dp"/>
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/greeting"
        android:textSize="30dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/greeting"
        android:textSize="30dp"/>

</LinearLayout>
```



How can we use String Array ?

Step1

Go to String.xml and add the below xml code inside the <resource></resource> tag.

String.xml

```
<string-array name="country">
   <item>India</item>
   <item>USA</item>
   <item>Africa</item>
   <item>Norway</item>
</string-array>
```

Activity_main.xml

```
<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:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity"
  android:orientation="vertical"
  android:padding="20dp">
   <Spinner
     android:id="@+id/spinner"
     android:layout_width="227dp"
     android:layout_height="97dp"
      android:entries="@array/country" />

<TextView
      android:id="@+id/tv"
      android:layout_width="285dp"
      android:layout_height="198dp"
      android:text="TextView"
      android:padding="20dp"
      android:layout_marginTop="50dp"/>
</LinearLayout>
```

colors.xml

The colors.xml file is responsible to **hold all the types of colors required for the application**. It may be primary brand color and its variants and secondary brand color and its variants. The colors help uphold the brand of the applications.

The colors need to be **defined in the hex code format**.

We can define some colors in a resource file, then apply those colors elsewhere in our app. By convention, colors are defined in a colors.xml file. Colors are considered "value

Add the below lines inside the **colors.xml** file.

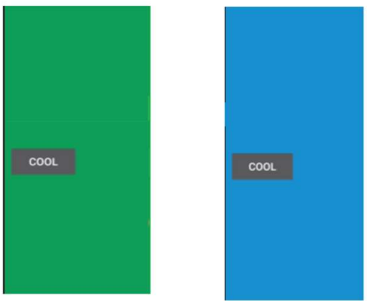
```
<color name="colorPrimary">#6200EE</color>
<color name="colorPrimaryDark">#3700B3</color>
<color name="colorAccent">#03DAC5</color>
<color name="green">#0F9D58</color>
<color name="cool">#188FCF</color>
<color name="warm">#F1D416</color>
```

activity_main.xml file

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/rlVar1"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@color/green"
   tools:context=".MainActivity">
  <Button
        android:id="@+id/btVar1"
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:padding="20dp"
        android:text="Cool"
        android:textSize="25dp"
        android:onClick=change()/>
RelativeLayout>
```

MainActivity.java file

```
public class MainActivity extends AppCompatActivity {
    @Override
 protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
 RELayout = findViewById(R.id.rlVar1);
public void change(View view)
{
          // set the color to relative layout
 RELayout.setBackgroundResource(R.color.cool);
 }
```



9.What is layout?Explain any one of layout with exampls

--Refer 7th ans

11.What is Fragment?Explain the procedure to create fragment.

- **Android Fragment** is the part of activity, it is also known as sub-activity.
- In other words, A Fragment in Android is a component which can be used over an activity to define an independent modular UI component attached to the activity

- There can be more than one fragment in an activity.
- Fragments represent multiple screen inside one activity.
- It functions independently, but as it is linked to the Activity, when an activity is destroyed, the fragment also gets destroyed.
- Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.
- Also, a fragment is a re-usable component, hence, a single fragment can be included in multiple activities, if required.
- Generally, fragments are used to create multi-pane UI in Android apps.
- A Fragment has it's own Layout for the UI (user interface), but we can even define a fragment without any layout, to implement a behaviors which has no user interface, more like a background service.

Creating Fragments

- A fragment represents a modular portion of the user interface within an activity.

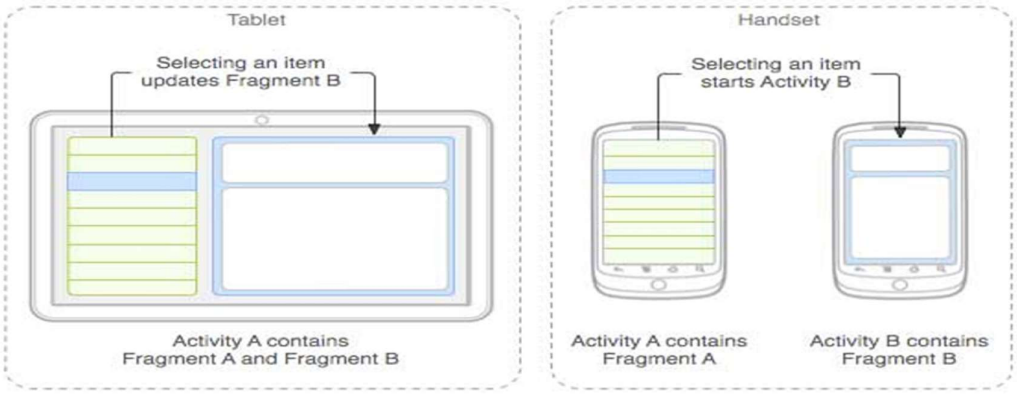
- A fragment is easier to reuse within activities and layouts. Android devices have a variety of screen sizes and densities.

- It simplifies the reuse of components in different layouts and their logic.

- We can also use fragments also to support different layouts for landscape and portrait orientation on a smartphone.

- we say a fragment is a kind of sub-activity. It represents a behavior or a portion of user interface in an Activity.
- We can combine multiple Fragments in Single Activity to build a multi panel UI and reuse a Fragment in multiple Activities.

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



12.Distinguish between Activity and Fragment?

Differences between Activity and Fragment

Activity	Fragment
Activity is an application component that gives a user interface where the user can interact.	The fragment is only part of an activity, it basically contributes its UI to that activity.
Activity is not dependent on fragment	Fragment is dependent on activity. It can't exist independently.
we need to mention all activity it in the manifest.xml file	Fragment is not required to mention in the manifest file
We can't create multi-screen UI without using fragment in an activity,	After using multiple fragments in a single activity, we can create a multi-screen UI.
Activity can exist without a Fragment	Fragment cannot be used without an Activity.
Creating a project using only Activity then it's difficult to manage	While Using fragments in the project, the project structure will be good and we can handle it easily.
Lifecycle methods are hosted by OS. The activity has its own life cycle.	Lifecycle methods in fragments are hosted by hosting the activity.
Activity is not lite weight.	The fragment is the lite weight.

Refer week experiments for 4 and 10 questions.