# Android

Android is an open-sourced operating system that is used on mobile devices, such as mobiles and tablets. The Android application executes within its own process and its own instance of Dalvik Virtual Machine(DVM) or Android RunTime(ART).

# Activity

Activity in java is a single screen that represents GUI(Graphical User Interface) with which users can interact in order to do something like dial the phone, view email, etc.

# Service

Service is an application component that facilitates an application to run in the background in order to perform long-running operations without user interaction. A service can run continuously in the background even if the application is closed or even after the user switches to another application.

# Activity vs Service

1. Activity is designed to run in the foreground. / Service is mainly designed to run in the background. Foreground services are also available.

2. Activity is used when the user interface is necessary. / Service is used when the user interface is not necessary.

3. Activity is dependent. / Service act independently.

# Bundle

Bundles are used to pass the required data between various Android activities. These are like HashMap that can take trivial data types.

# Adapter

An adapter in Android acts as a bridge between an AdapterView and the underlying data for that view. The adapter holds the data and sends the data to the adapter view, the view can take the data from the adapter view and shows the data on different views like a spinner, list view, grid view, etc.

# DDMS

1. Port forwarding services.

2. Thread and heap information.

3. Logcat.

4. Screen capture on the device.

5. Network traffic tracking.

6. Incoming call and SMS spoofing.

7. Location data spoofing.

# Activity Lifecycle

1. OnCreate(): It is called when activity is created. Using this, the views are created and data is collected from bundles.

2. OnStart(): It is called if the activity is becoming visible to the user. It may be succeeded by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.

3. OnResume(): It is called when the activity will start an interaction with the user.

4. OnPause(): This is called when the activity is moving to the background but hasn’t been killed yet.

5. OnStop(): This is called when an activity is no longer visible to the user.

6. OnDestroy(): This is called when the activity is finished or destroyed.

7. OnRestart(): This is called after the activity has been stopped, prior to it being started again.

https://s3.ap-south-1.amazonaws.com/myinterviewtrainer-domestic/public\_assets/assets/000/000/441/original/Life\_Cycle\_of\_Android.png

Differentiate:

onCreate() is the first method that’s invoked when an activity is launched for the first time. onStart() is invoked after onCreate() has completed it’s task. onResume() is called after onStart() has completed. When an activity leaves its foreground (probably for a smaller duration such as standby/sleep) onPause() is invoked followed by onStop()(when the activity is not visible. eg. some other application is launched). onDestroy() is called when the activity or application is killed.

Essentially the lifecycle methods are divided into three layers of duration :

1. onCreate() and onDestroy() are present during the entire duration of the activity

2. onStart() and onStop() are present while the activity is visible

3. onResume() and onPause() are present while the activity is in foreground

# Sensor

Android-based devices have a collection of built-in sensors in them, which measure certain parameters like motion, orientation, and many more through their high accuracy. The sensors can be both hardware and software based on nature. There are three prominent categories of sensors in Android devices.

They are:

1. Position Sensor: It is used for measuring the physical position of the Android device. This has orientation sensors and magnetometers.

2. Motion Sensors: These sensors consist of gravity, rotational activity, and acceleration sensors which measure the rotation of the device or the acceleration, etc.

3. Environmental Sensor: It includes sensors that measure temperature, humidity, pressure, and other environmental factors.

# dialog boxes

1. AlertDialog:

The AlertDialog supports 0-3 buttons, along with a list of selectable items such as checkboxes and radio buttons.

It is used when you want to ask the user about taking a decision between yes or no in response to any particular action taken by the user, by remaining in the same activity and without changing the screen.

2. DatePickerDialog:

It is used for selecting the date by the user.

3. TimePickerDialog:

Used for selecting the time by the user.

4. ProgressDialog:

It is an extension of the AlertDialog and is used to display a progress bar. It also supports the addition of buttons.

This class was deprecated in API level 26 because it prevents the user from interacting with the application. Instead of this class, we can use a progress indicator such as ProgressBar, which can be embedded in the user interface of your application.

# AndroidManifest.xml

1. The AndroidManifest.xml file contains information regarding the application that the Android system must know before the codes can be executed.

2. This file is essential in every Android application.

3. It is declared in the root directory.

4. This file performs several tasks such as:

4-1. Providing a unique name to the java package.

4-2. Describing various components of the application such as activity, services, and many more.

4-3. Defining the classes which will implement these components.

# intent

An intent is a messaging object that is used to request an action from other components of an application. It can also be used to launch an activity, send SMS, send an email, display a web page, etc.

It shows notification messages to the user from within an Android-enabled device. It alerts the user of a particular state that occurred.

There are two types of intents in Android:

1. Implicit Intent- Used to invoke the system components.

2. Explicit Intent- Used to invoke the activity class.

# Toast

Toast is a message that pops up on the screen. It is used to display the message regarding the status of the operation initiated by the user and covers only the expanse of space required for the message while the user’s recent activity remains visible and interactive.

Toast notification automatically fades in and out and it does not accept interaction events.

# context

The context in Android is the context of the current state of the application or object. The context comes with services like giving access to databases and preferences, resolving resources, and more.

There are two types of context. They are:

1. Activity context

1-1. This activity context is attached to the lifecycle of an activity.

1-2. The activity context can be used when you are passing the context in the scope of an activity or you need the context whose lifecycle is attached to the context of the activity.

2. Application context:

2-1. This application context is attached to the lifecycle of an application.

2-2. The application context should be used where you need a context whose lifecycle is separate from the current context or when you are passing a context beyond the scope of activity.

# Intent - Implicit vs Explicit

Explicit Intent:

An Explicit Intent is where you inform the system about which activity should handle this intent. Here target component is defined directly in the intent.

Implicit Intent:

An Implicit Intent permits you to declare the action you want to carry out. Further, the Android system will check which components are registered to handle that specific action based on intent data. Here target component is not defined in the intent.

# launch mode

1. Standard:

This launch mode generates an activity’s new instance in the task from which it originated.

It is possible to create several instances for the same activity.

2. SingleTop:

This launch mode is similar to the Standard launch mode except if there exists an activity’s previous instance on the top of the stack, then a new instance will not be created.

But the intent will be sent to the activity’s existing instance.

3. SingleTask:

This launch mode will create a new task and push a new instance to the task as the root.

4. SingleInstance:

This launch mode is similar to the SingleTask launch mode. But the system doesn’t support launching any new activities in the same task.

In a situation where the new activity is launched, it is launched in a separate task.

# Container

Containers carry objects and widgets together, based on which specific items are required and in what particular arrangement is needed. Containers may hold labels, buttons, fields, or even child containers, etc. For example, if you want a form with fields on the left and labels on the right, you will need a container. If you want the OK and Cancel buttons to be below the rest of the form, next to one another, and flush to the right side of the screen, you will need a container. If you have several widgets, you will need a container to have a root element to place the widgets inside.

Android provides a collection of view classes that serve as containers for views. These container classes are called layouts, which are defined in the form of XML files that cannot be changed by our code during execution. The layout managers provided by Android SDK are LinearLayout, RelativeLayout, FrameLayout, AbsoluteLayout, GridLayout, and TableLayout.

# broadcast receiver

A broadcast receiver is a mechanism used for listening to system-level events like listening for incoming calls, SMS, etc. by the host application. It is implemented as a subclass of BroadcastReceiver class and each message is broadcasted as an intent object.

# Serializable vs Parcelable

While developing applications usually it needs to transfer data from one activity to another. This data needs to be added into a corresponding intent object. Some additional actions are required to make the data suitable for transfer. For doing that the object should be either serializable or parcelable.

1. Serializable:

1-1. Serializable is a standard Java interface. In this approach, you simply mark a class Serializable by implementing the interface and java will automatically serialize it.

1-2. Reflection is used during the process and many additional objects are created. This leads to plenty of garbage collection and poor performance.

2. Parcelable:

2-1. Parcelable is an Android-specific interface. In this approach, you implement the serialization yourself.

2-2. Reflection is not used during this process and hence no garbage is created.

2-3. Parcelable is far more efficient than Serializable since it gets around some problems with the default Java serialization scheme. Also, it is faster because it is optimized for usage on the development of Android, and shows better results.

# SQLite

SQLite is the open-source relational database used in Android. The SQLite engine is serverless, transactional, and also self-contained. Instead of the client-server relationship of most database management systems, the SQLite engine is integrally linked with the application. The library can be called dynamically and it can make use of simple function calls that reduce latency in database access.

# sqlite

1. Service is an application component that facilitates an application to run in the background in order to perform long-running operations without user interaction. / A Thread is a concurrent unit of execution.

2. Service exposes few functionalities to other applications by calling Context.bindService(). / For Thread, Google has brought in handlers and loopers into threads.

3. When an application is killed, service is not killed. / When an application is killed, the thread is killed.

# sqlite

1. Service is an application component that facilitates an application to run in the background in order to perform long-running operations without user interaction. / A Thread is a concurrent unit of execution.

2. Service exposes few functionalities to other applications by calling Context.bindService(). / For Thread, Google has brought in handlers and loopers into threads.

3. When an application is killed, service is not killed. / When an application is killed, the thread is killed.

# Content provider

Content provider is one of the primary building blocks of Android applications, which manages access to a central repository of data. It acts as a standard interface that connects data in one process with code running in another process. So it can be used to share the data between different applications.

They are responsible for encapsulating the data and providing mechanisms for defining data security. It is implemented as a subclass of ContentProviderclass and must implement a set of APIs that will enable other applications to perform transactions.

# compileSdkVersion vs targetSdkVersion

1. compileSdkVersion:

1-1. The compileSdkVersion is the version of API the application is compiled against. You can use Android API features involved in that version of the API (as well as all previous versions).

1-2. For example, if you try and use API 15 features but set compileSdkVersion to 14, you will get a compilation error. If you set compileSdkVersion to 15 you can still run the app on an API 14 device as long as your app’s execution paths do not attempt to invoke any APIs specific to API 15.

2. targetSdkVersion:

2-1. The targetSdkVersion indicates that you have tested your app on (presumably up to and including) the version you specify. This is like a certification or sign-off you are giving the Android OS as a hint to how it should handle your application in terms of OS features.

2-2. For example, setting the targetSdkVersion value to “11” or higher permits the system to apply a new default theme (Holo) to the application when running on Android 3.0 or higher. It also disables screen compatibility mode when running on larger screens (because support for API level 11 implicitly supports larger screens).

# targetSdkVersion vs compileSdkVersion

1. compileSdkVersion:

1-1. The compileSdkVersion is the version of API the application is compiled against. You can use Android API features involved in that version of the API (as well as all previous versions).

1-2. For example, if you try and use API 15 features but set compileSdkVersion to 14, you will get a compilation error. If you set compileSdkVersion to 15 you can still run the app on an API 14 device as long as your app’s execution paths do not attempt to invoke any APIs specific to API 15.

2. targetSdkVersion:

2-1. The targetSdkVersion indicates that you have tested your app on (presumably up to and including) the version you specify. This is like a certification or sign-off you are giving the Android OS as a hint to how it should handle your application in terms of OS features.

2-2. For example, setting the targetSdkVersion value to “11” or higher permits the system to apply a new default theme (Holo) to the application when running on Android 3.0 or higher. It also disables screen compatibility mode when running on larger screens (because support for API level 11 implicitly supports larger screens).

# Sensor class

1. Sensor class: This class helps you to create an instance of a specific sensor. It provides methods that let you determine a sensor’s capabilities.

2. SensorManager class: This class is used to create an instance of the sensor service. It provides methods to access and list sensors, to register and unregister sensor listeners, etc.

3. SensorEvent class: This Java class is used to create a sensor event object. It provides information about the sensor event including raw sensor data, the accuracy of data, type of sensor, timestamp of event, etc.

4. SensorEventListener interface: This interface is used to create two callback methods that receive sensor event notifications when sensor value changes or when sensor accuracy changes. Those two methods are onAccuracyChanged which is called when sensor accuracy is changed and

onSensorChanged which is called when sensor values are changed.

# JobSchedular

The JobSchedular API is used for scheduling different types of jobs against the framework that will be executed in your app’s own process. This allows your application to perform the given task while being considerate of the device’s battery at the cost of timing control.

The JobScheduler supports batch scheduling of jobs. The Android system can combine jobs for reducing battery consumption. JobManager automatically handles the network unreliability so it makes handling uploads easier.

Here is some example of the situation where you would use this job scheduler:

1. Tasks that should be done when the device is connected to a power supply.

2. Tasks that require a Wi-Fi connection or network access.

3. Tasks that should run on a regular basis as batch where the timing is not critical.

# Android application Architecture

1. Services: Used to perform background functionalities.

2. Intent: Used to perform the interconnection between activities and the data passing mechanism.

3. Resource Externalization: strings and graphics.

4. Notification: light, sound, icon, notification, dialog box and toast.

5. Content Providers: It will share the data between applications.

# storage in Android

1. Shared Preferences

2.Internal Storage

3. External Storage

4. SQLite Databases

5. Network Connection

# fragment

The fragment is a part of Activity by which we can display multiple screens on one activity.

# Android SDK

The Google Android SDK is a toolset which is used by developers to write apps on Android-enabled devices. It contains a graphical interface that emulates an Android-driven handheld environment and allows them to test and debug their codes.

# NDK

NDK stands for Native Development Kit. By using NDK, you can develop a part of an app using native language such as C/C++ to boost the performance.

# View Group

View Group is a collection of views and other child views. It is an invisible part and the base class for layouts.

# application Widgets in android

Application widgets are miniature application views that can be embedded in other applications and receive periodic updates.

# singleton

A singleton class is a class which can create only an object that can be shared by all other classes.

# sleep mode

In sleep mode, CPU is slept and doesn't accept any commands from android device except Radio interface layer and alarm.

# drawable folder

In Android, a drawable folder is compiled a visual resource that can use as a background, banners, icons, splash screen, etc.

# exceptions in Android

1. Inflate Exception

2. Surface.OutOfResourceException

3. SurfaceHolder.BadSurfaceTypeException

4. WindowManager.BadTokenException

# Activity vs AppCompatActivity

AppCompatActivity provides native ActionBar support that is consistent across the application.

Also, it provides backward compatibility for other material design components till SDK version 7(ActionBar was natively available since SDK 11).

Extending an Activity doesn’t provide any of these.

Note: Since SDK 21 every activity by default, extends AppCompatActivity.

# Activity, AppCompatActivity, FragmentActivity and ActionBarActivity

Activity is the base class.

FragmentActivity extends Activity.

AppCompatActivity extends FragmentActivity.

ActionBarActivity extends AppCompatActivity.

FragmentActivity is used for fragments.

Since the build version 22.1.0 of the support library, ActionBarActivity is deprecated.

It was the base class of appcompat-v7.

At present, AppCompatActivity is the base class of the support library.

It has come up with many new features like ToolBar, tinted widgets, material design color pallets etc.

# Android Support Library

The android platform supports a wide variety of the versions and devices to choose from. With the release of every new version, new Android APIs are added and evolved. To make these new Android APIs available to users on older devices the Android Support Library was designed. Android Support Library provides developers with newer APIs that are compatible on older framework releases.

1. Compatibility Libraries: These focus on back porting features so that older frameworks can take advantage of newer releases. The major libraries include v4 and v7-appcompat. v4 includes classes like DrawerLayout and ViewPager while appcompat-v7 provides classes for support ActionBar and ToolBar.

2. Component Libraries: These include libraries of certain modules that don’t depend on other support library dependencies. They can be easily added or removed. Examples include v7-recyclerview and v7-cardview.

3. Miscellaneous libraries: The Android support libraries consists of few other libraries such as v8 which provides support for RenderScript, annotations for supporting annotations like @NonNull.

# onPause() only scenario

Create and launch a new activity which obscures the current activity partially. This can be done by defining the layout\_width and layout\_height to partially cover the screen. This would keep the first activity visible but not in the foreground. Example: define the layout\_width and layout\_height as 200dp each.

# activity- screen rotate

When the screen is rotated, the current instance of the activity is destroyed a new instance of the Activity is created in the new orientation. The onRestart() method is invoked first when a screen is rotated. The other lifecycle methods get invoked in the similar flow as they were when the activity was first created.

How to prevent the data from reloading and resetting when the screen is rotated?

The most basic approach is to add an element attribute tag `android:configChanges` inside the activity tag in the AndroidManifest.xml as shown below.

<activity android:name="".MainActivity""

android:configChanges=""orientation|screenSize"">

<intent-filter>

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

<category android:name=""android.intent.category.LAUNCHER"" />

</intent-filter>

</activity>

In general, the configChanges for any activity are defined as

android:configChanges=""orientation|screenSize|keyboardHidden""

The `keyboardHidden` configuration is to prevent the keyboard from resetting if it's pulled out.

`android:configChanges` is not the recommended way by Google. Though it's the simplest way to use, it comes with its own share of drawbacks. First, the common perception that android:configChanges = ""orientation"" will magically retain the data is a complete misinterpretation. The orientation changes can occur from a number of other events such as changing the default language can trigger a configuration change and destroy and recreate the activity. Second, the activity can restart itself if it's in the background and Android decides to free up its heap memory by killing it. When the application returns to the foreground it'll restart it's data to the original state and the user may not like that. A better alternative of `android:configChanges` is; Saving the current state of the activity when it's being destroyed and restoring the valuable data when it's restarted can be done by overriding the methods `onSaveInstanceState()` and `onRestoreInstanceState()` of the activity class.

# onSaveInstanceState() vs onRestoreInstanceState()

In general the onSaveInstanceState() is invoked after onPause() and before the onStop(). But the API documentation explicitly states that the onSaveInstanceState( ) method will be called before onStop() but makes no guarantees it will be called before or after onPause(). The onRestoreInstanceState() is called after onStart() is invoked. The onRestoreInstanceState() method is invoked only when the activity was killed before. If the activity is NOT killed the onSaveInstanceState() is NOT called. When the activity is being destroyed, the onSaveInstanceState() gets invoked. The onSaveInstanceState contains a Bundle parameter. The data to be saved is stored in the bundle object in the form of a HashMap. The bundle object is like a custom HashMap object. The data is retrieved in the onRestoreInstanceState() method using the keys.

# sqlite

The screen orientation can be fixed by adding the attribute `android:screenOrientation=""portrait""` or `android:screenOrientation=""landscape""` in the activity tag. To keep the screen always on for a particular screen add the `android:keepScreenOn=""true""` in the root tag of the activity layout.

# screen orientation fixed

public class MainActivity extends AppCompatActivity {

public void restartActivity() {

MainActivity.this.recreate();

}

}

# intent usages

Android Intents are used to

1. start an activity - startActivity(intent)

2. start a service - startService(intent)

3. deliver a broadcast - sendBroadcast(intent)