# 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.

For example, the Facebook start page where you enter your email/phone number and password to log in acts as an activity.

# activity lifecycle

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

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.

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

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

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

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

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

# activity vs service

Activities can be terminated or closed anytime the user wishes. On the other hand, services are designed to run in the background, and they can act independently.

Most of the services run continuously, irrespective of whether there are certain or no activities being executed.

Activities are designed to run in the foreground. / Services are mainly designed to run in the background. Foreground services are also available.

Activities are used when the user interface is necessary. / Services are used when the user interface is not necessary.

Activities are dependent. / Services act independently.

# 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.

# androidmanifest.xml

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

This file is essential in every Android application.

It is declared in the root directory.

This file performs providing a unique name to the java package, describing various components of the application such as activity, services, and many more, defining the classes which will implement these components.

# 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.

# bundle

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

# compilesdkversion vs targetsdkversion

compileSdkVersion:

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).

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.

targetSdkVersion:

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.

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).

# 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.

# 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.

# 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.

Activity context is attached to the lifecycle of an activity.

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.

Application context is attached to the lifecycle of an application.

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

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:

Implicit Intent is used to invoke the system components. (Map geo location, dial a call, launch website)

Explicit Intent is used to invoke the activity class. (Start Broadcast Receivers, launch activiy, start background service)

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.

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.

# jobscheduler

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:

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

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

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

# launch mode

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.

For Example, suppose our current stack is A -> B -> C. Now, if we launch activity B again with the “standard” launch mode, then the new stack will be A -> B -> C -> B.

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.

For example, suppose our current stack is A -> B -> C. Now, if we launch the activity B again with “singleTop” launch mode,then the new stack will be A -> B -> C -> B.

Consider another example, where the current stack is A -> B -> C. Now, if we launch activity C again with the “singleTop” launch mode, then the stack will remain the same i.e., A -> B -> C. The intent will be passed to the onNewIntent() method.

SingleTask:

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

For example, suppose our current stack is A -> B -> C -> D. Now, if we launch activity B again with the “singleTask” launch mode, then the new stack will be A -> B. Here, a callback has been received on the old instance and C and D activities are destroyed.

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.

For example, Suppose our current stack is A -> B -> C. Now, if we launch the activity D with the “singleInstance” launch mode, then there will be two stacks:

A -> B -> C

D, If you call activity E, then it will be added to the first stack.

A -> B -> C -> E

D

Again if you Call the activity D, then it will call the same activity from the 2nd stack and pass the intent to onNewIntent()."

# sdk

The Google Android SDK(Android software development kit) is a toolset used by developers to write applications on Android-enabled devices.

Android Emulator - Android Emulator is a software application that simulates Android devices on computer so that you can test the application on a variety of devices and Android API levels without having each physical device.

DDMS(Dalvik Debug Monitoring Services) - It is a debugging tool from the SDK which provides services like message formation, call spoofing, capturing screenshots, etc.

ADB(Android Debug Bridge) - It is a command-line tool used to allow and control communication with the emulator instance.

AAPT(Android Asset Packaging Tool) - It is a build tool that gives the ability to developers to view, create, and update ZIP-compatible archives (zip, jar, and apk)."

# sensor

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

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

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

# sensor class

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.

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.

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.

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 void onAccuracyChanged(Sensor sensor, int accuracy) which is called when sensor accuracy is changed and

void onSensorChanged(SensorEvent event) which is called when sensor values are changed.

# 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.

Serializable:

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.

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

Parcelable:

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

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

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.

# 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.

# service vs thread

>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.

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

>When an application is killed, service is not killed. When an application is killed, the thread is killed."

# 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.

# 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.