Name : Mujahid Ullah

Subject: Android Development

Submitted To Sir Junaid

Date : 25-Dec-20

---

1. Android activity life cycle

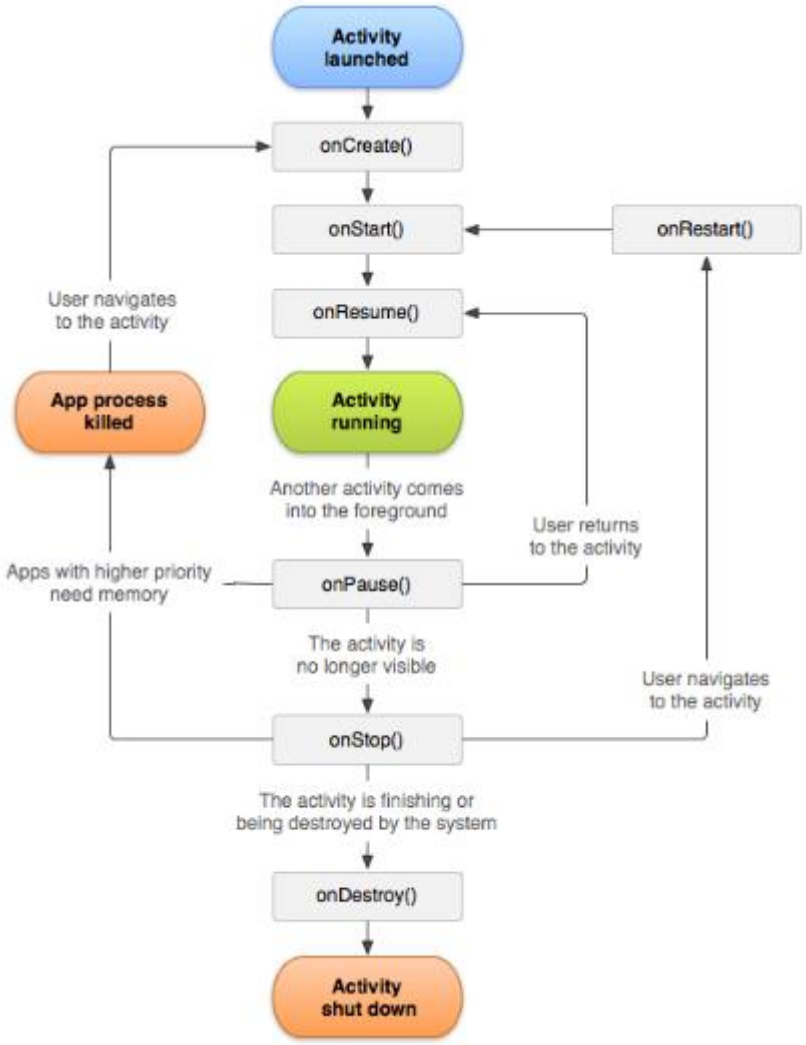
Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

An activity is the single screen in android. It is like window or frame of Java.By the help of activity, you can place all your UI components or widgets in a single screen.

The 7 lifecycle method of Activity describes how activity will behave at different states.

Methods	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.

Diagram Which Which Android Activity Life Cycle



Source Code Of Android Activity Life Cycle

```
package example.javatpoint.com.activitylifecycle;


import android.app.Activity;

import android.os.Bundle;
```

```
import android.util.Log;


public class MainActivity extends Activity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
```

```
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}
```

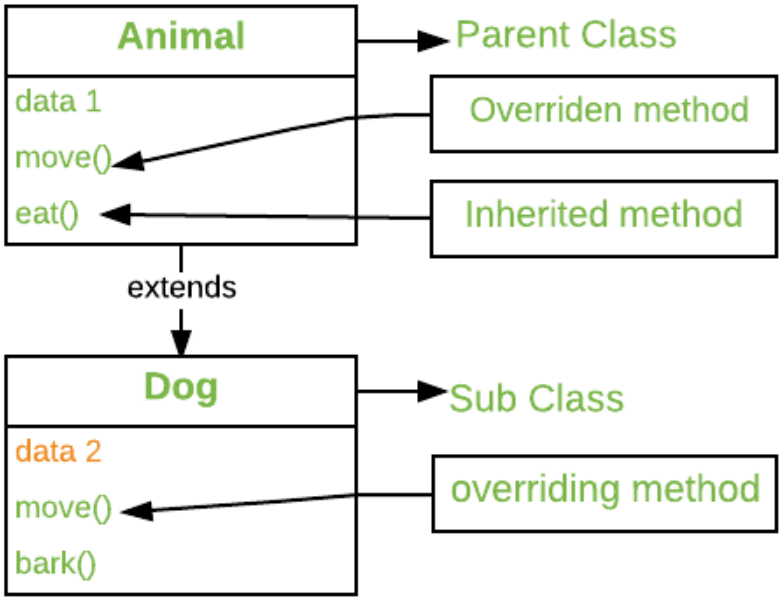
Ref: https://www.javatpoint.com/android-life-cycle-of-activity

2. Function overriding in java

Overriding in Java:

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature, and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Diagram



Method overriding is one of the way by which java achieve Run Time Polymorphism.The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

Source Code :

```
// A Simple Java program to demonstrate

// method overriding in java
```

```
// Base Class
class Parent {
	void show()
	{
		System.out.println("Parent's show()");
	}
}

// Inherited class
class Child extends Parent {
	// This method overrides show() of Parent
	@Override
	void show()
	{
		System.out.println("Child's show()");
	}
}

// Driver class
class Main {
	public static void main(String[] args)
	{
		// If a Parent type reference refers
		// to a Parent object, then Parent's
```

```
        // show is called
        Parent obj1 = new Parent();
        obj1.show();

        // If a Parent type reference refers
        // to a Child object Child's show()
        // is called. This is called RUN TIME
        // POLYMORPHISM.
        Parent obj2 = new Child();
        obj2.show();
    }
}
```

3. Activity INTER-APPLICATION COMMUNICATION IN ANDROID

Some apps attempt to implement IPC using traditional Linux techniques such as network sockets and shared files. However, you should instead use Android system functionality for IPC such as Intent, Binder or Messenger with a Service, and BroadcastReceiver. The Android IPC mechanisms allow you to verify the identity of the application connecting to your IPC and set security policy for each IPC mechanism.

Hence we will limit our discussions the mechanisms evangelised in the official android documentation. We shall also only discuss IPC within the same application and not IPC amongst different applications.

Using Intents

Intents and Intent Filters

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:

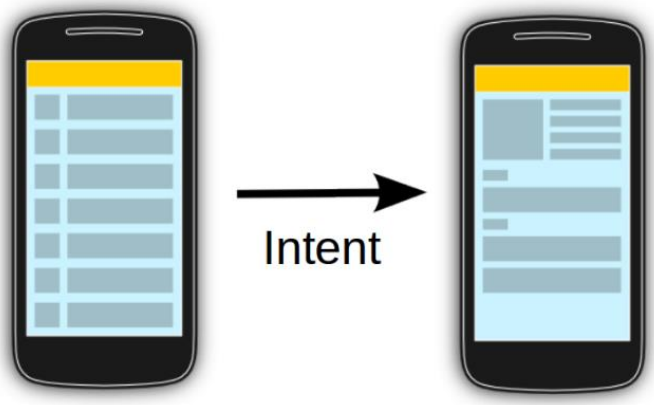
I. Starting an activity

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.

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.

Process:

To start an activity, use the method startActivity(intent). This method is defined on the Context object which Activity extends.



The following code demonstrates how you can start another activity via an intent.

Example

```
// Start the activity connect to the specified class

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

startActivity(i);
```

Activities which are started by other Android activities are called sub-activities. This wording makes it easier to describe which activity is meant.

To start a services via intents, use the startService(Intent) method call.

II. 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 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.

Life Cycle of Android Service

There can be two forms of a service.The lifecycle of service can follow two different paths: started or bound.

1) Started Service

A service is started when component (like activity) calls startService() method, now it runs in the background indefinitely. It is stopped by stopService() method. The service can stop itself by calling the stopSelf() method.

2) Bound Service

A service is bound when another component (e.g. client) calls bindService() method. The client can unbind the service by calling the unbindService() method.

The service cannot be stopped until all clients unbind the service.

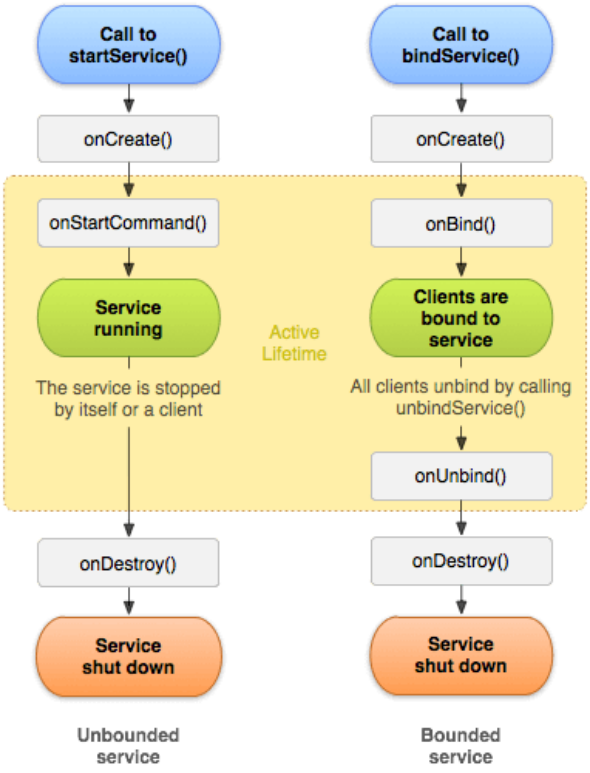


Figure Show Life Cycle of Android Service

Source Code Example

```
public class MyService extends Service {
    MediaPlayer myPlayer;
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        Toast.makeText(this, "Service Created", Toast.LENGTH_LONG).show();

        myPlayer = MediaPlayer.create(this, R.raw.sun);
        myPlayer.setLooping(false); // Set looping
    }
```

```
    @Override
    public void onStart(Intent intent, int startid) {
        Toast.makeText(this, "Service Started", Toast.LENGTH_LONG).show();
        myPlayer.start();
    }
    @Override
    public void onDestroy() {
        Toast.makeText(this, "Service Stopped", Toast.LENGTH_LONG).show();
        myPlayer.stop();
    }
}
```

III. 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().

The rest of this page explains how intents work and how to use them. For related information, see Interacting with Other Apps and Sharing Content.

Example

Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern. These broadcasts are sent when an event of interest occurs. For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging. Apps can also send custom broadcasts, for example, to notify other apps of something that they might be interested in (for example, some new data has been downloaded).

Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.