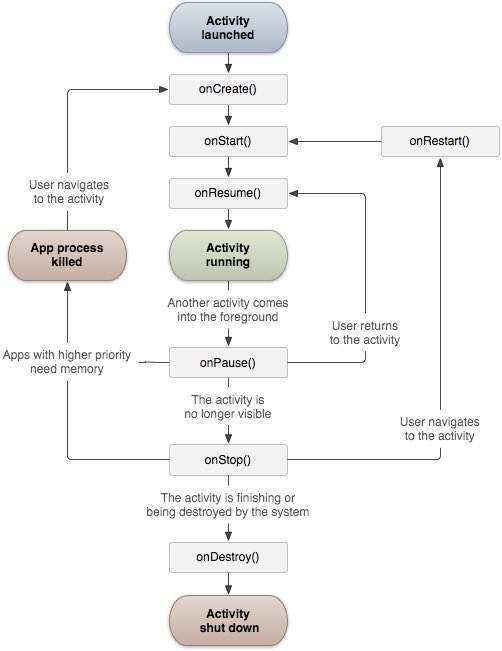
An activity represents a single screen with a user interface just like window or frame of Java. Android activity is the subclass of ContextThemeWrapper class.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function.

Very similar way, Android system initiates its program with in an **Activity** starting with a call on ***onCreate()* callback method**.

There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram:



The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks (**which function is called**) methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

|  |  |
| --- | --- |
| **Sr.No** | **Callback & Description** |
| 1 | **onCreate()**  This is the first callback and called when the activity is first created. |
| 2 | **onStart()**  This callback is called when the activity becomes visible to the user. |
| 3 | **onResume()**  This is called when the user starts interacting with the application. |
| 4 | **onPause()**  The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed. |
| 5 | **onStop()**  This callback is called when the activity is no longer visible. |
| 6 | **onDestroy()**  This callback is called before the activity is destroyed by the system. |
| 7 | **onRestart()**  This callback is called when the activity restarts after stopping it. |

**Example**

This example will take you through simple steps to show Android application activity life cycle. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android studio to create an Android application and name it as *HelloWorld* under a package *com.example.helloworld* as explained in the *Hello World Example* chapter. |
| 2 | Modify main activity file *MainActivity.java* as explained below. Keep rest of the files unchanged. |
| 3 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **src/com.example.helloworld/MainActivity.java**. This file includes each of the fundamental life cycle methods. The **Log.d()** method has been used to generate log messages −

package com.example.helloworld;

import android.os.Bundle;

import android.app.Activity;

import android.util.Log;

public class MainActivity extends Activity {

String msg = "Android : ";

/\*\* Called when the activity is first created. \*/

@Override

public void **onCreate**(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Log.d(msg, "The onCreate() event");

}

/\*\* Called when the activity is about to become visible. \*/

@Override

protected void **onStart()** {

super.onStart();

Log.d(msg, "The onStart() event");

}

/\*\* Called when the activity has become visible. \*/

@Override

protected void **onResume()** {

super.onResume();

Log.d(msg, "The onResume() event");

}

/\*\* Called when another activity is taking focus. \*/

@Override

protected void **onPause()** {

super.onPause();

Log.d(msg, "The onPause() event");

}

/\*\* Called when the activity is no longer visible. \*/

@Override

protected void **onStop()** {

super.onStop();

Log.d(msg, "The onStop() event");

}

/\*\* Called just before the activity is destroyed. \*/

@Override

public void **onDestroy()** {

super.onDestroy();

Log.d(msg, "The onDestroy() event");

}

}

An activity class loads all the UI component using the XML file available in *res/layout* folder of the project. Following statement loads UI components from *res/layout/activity\_main.xml file*:

setContentView(R.layout.activity\_main);

An application can have one or more activities without any restrictions. Every activity you define for your application must be declared in your *AndroidManifest.xml* file and the main activity for your app must be declared in the manifest with an <intent-filter> that includes the MAIN action and LAUNCHER category as follows:

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

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

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

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

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

</intent-filter>

</activity>

</application>

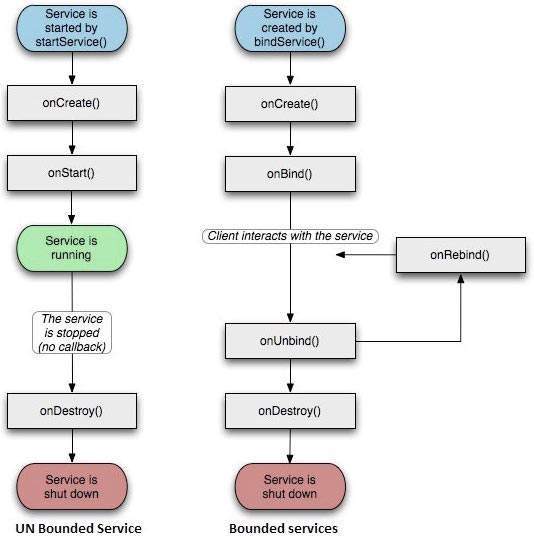
</manifest>

If either the MAIN action or LAUNCHER category are not declared for one of your activities, then your app icon will not appear in the Home screen's list of apps.

A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states −

|  |  |
| --- | --- |
| **Sr.No.** | **State & Description** |
| 1 | **Started**  A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed. |
| 2 | **Bound**  A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC). |

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService(): *(image courtesy : android.com )*



To create an service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

|  |  |
| --- | --- |
| **Sr.No.** | **Callback & Description** |
| 1 | **onStartCommand()**  The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()* methods. |
| 2 | **onBind()**  The system calls this method when another component wants to bind with the service by calling *bindService()*. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an *IBinder* object. You must always implement this method, but if you don't want to allow binding, then you should return *null*. |
| 3 | **onUnbind()**  The system calls this method when all clients have disconnected from a particular interface published by the service. |
| 4 | **onRebind()**  The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its *onUnbind(Intent)*. |
| 5 | **onCreate()**  The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up. |
| 6 | **onDestroy()**  The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc. |

The following skeleton service demonstrates each of the life cycle methods −

package com.tutorialspoint;

import android.app.Service;

import android.os.IBinder;

import android.content.Intent;

import android.os.Bundle;

public class HelloService extends Service {

/\*\* indicates how to behave if the service is killed \*/

int mStartMode;

/\*\* interface for clients that bind \*/

IBinder mBinder;

/\*\* indicates whether onRebind should be used \*/

boolean mAllowRebind;

/\*\* Called when the service is being created. \*/

@Override

public void onCreate() {

}

/\*\* The service is starting, due to a call to startService() \*/

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

return mStartMode;

}

/\*\* A client is binding to the service with bindService() \*/

@Override

public IBinder onBind(Intent intent) {

return mBinder;

}

/\*\* Called when all clients have unbound with unbindService() \*/

@Override

public boolean onUnbind(Intent intent) {

return mAllowRebind;

}

/\*\* Called when a client is binding to the service with bindService()\*/

@Override

public void onRebind(Intent intent) {

}

/\*\* Called when The service is no longer used and is being destroyed \*/

@Override

public void onDestroy() {

}

}

Example

This example will take you through simple steps to show how to create your own Android Service. Follow the following steps to modify the Android application we created in *Hello World Example* chapter −

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | You will use Android StudioIDE to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication* as explained in the *Hello World Example* chapter. |
| 2 | Modify main activity file *MainActivity.java* to add *startService()* and *stopService()* methods. |
| 3 | Create a new java file *MyService.java* under the package *com.example.My Application*. This file will have implementation of Android service related methods. |
| 4 | Define your service in *AndroidManifest.xml* file using <service.../> tag. An application can have one or more services without any restrictions. |
| 5 | Modify the default content of *res/layout/activity\_main.xml* file to include two buttons in linear layout. |
| 6 | No need to change any constants in *res/values/strings.xml* file. Android studio take care of string values |
| 7 | Run the application to launch Android emulator and verify the result of the changes done in the application. |

Following is the content of the modified main activity file **MainActivity.java**. This file can include each of the fundamental life cycle methods. We have added *startService()* and *stopService()* methods to start and stop the service.

package com.example.tutorialspoint7.myapplication;

import android.content.Intent;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.os.Bundle;

import android.app.Activity;

import android.util.Log;

import android.view.View;

public class MainActivity extends Activity {

String msg = "Android : ";

/\*\* Called when the activity is first created. \*/

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Log.d(msg, "The onCreate() event");

}

public void startService(View view) {

startService(new Intent(getBaseContext(), MyService.class));

}

// Method to stop the service

public void stopService(View view) {

stopService(new Intent(getBaseContext(), MyService.class));

}

}

Following is the content of **MyService.java**. This file can have implementation of one or more methods associated with Service based on requirements. For now we are going to implement only two methods *onStartCommand()* and *onDestroy()* −

package com.example.tutorialspoint7.myapplication;

import android.app.Service;

import android.content.Intent;

import android.os.IBinder;

import android.support.annotation.Nullable;

import android.widget.Toast;

/\*\*

\* Created by TutorialsPoint7 on 8/23/2016.

\*/

public class MyService extends Service {

@Nullable

@Override

public IBinder onBind(Intent intent) {

return null;

}

@Override

public int onStartCommand(Intent intent, int flags, int startId) {

// Let it continue running until it is stopped.

Toast.makeText(this, "Service Started", Toast.LENGTH\_LONG).show();

return START\_STICKY;

}

@Override

public void onDestroy() {

super.onDestroy();

Toast.makeText(this, "Service Destroyed", Toast.LENGTH\_LONG).show();

}

}

Following will the modified content of *AndroidManifest.xml* file. Here we have added <service.../> tag to include our service −

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

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

package="com.example.tutorialspoint7.myapplication">

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

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

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

</intent-filter>

</activity>

<service android:name=".MyService" />

</application>

</manifest>

Following will be the content of **res/layout/activity\_main.xml** file to include two buttons −

<RelativeLayout 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:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

android:paddingBottom="@dimen/activity\_vertical\_margin" tools:context=".MainActivity">

<TextView

android:id="@+id/textView1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Example of services"

android:layout\_alignParentTop="true"

android:layout\_centerHorizontal="true"

android:textSize="30dp" />

<TextView

android:id="@+id/textView2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Tutorials point "

android:textColor="#ff87ff09"

android:textSize="30dp"

android:layout\_above="@+id/imageButton"

android:layout\_centerHorizontal="true"

android:layout\_marginBottom="40dp" />

<ImageButton

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/imageButton"

android:src="@drawable/abc"

android:layout\_centerVertical="true"

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:id="@+id/button2"

android:text="Start Services"

android:onClick="startService"

android:layout\_below="@+id/imageButton"

android:layout\_centerHorizontal="true" />

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Stop Services"

android:id="@+id/button"

android:onClick="stopService"

android:layout\_below="@+id/button2"

android:layout\_alignLeft="@+id/button2"

android:layout\_alignStart="@+id/button2"

android:layout\_alignRight="@+id/button2"

android:layout\_alignEnd="@+id/button2" />

</RelativeLayout>