CHAPTER

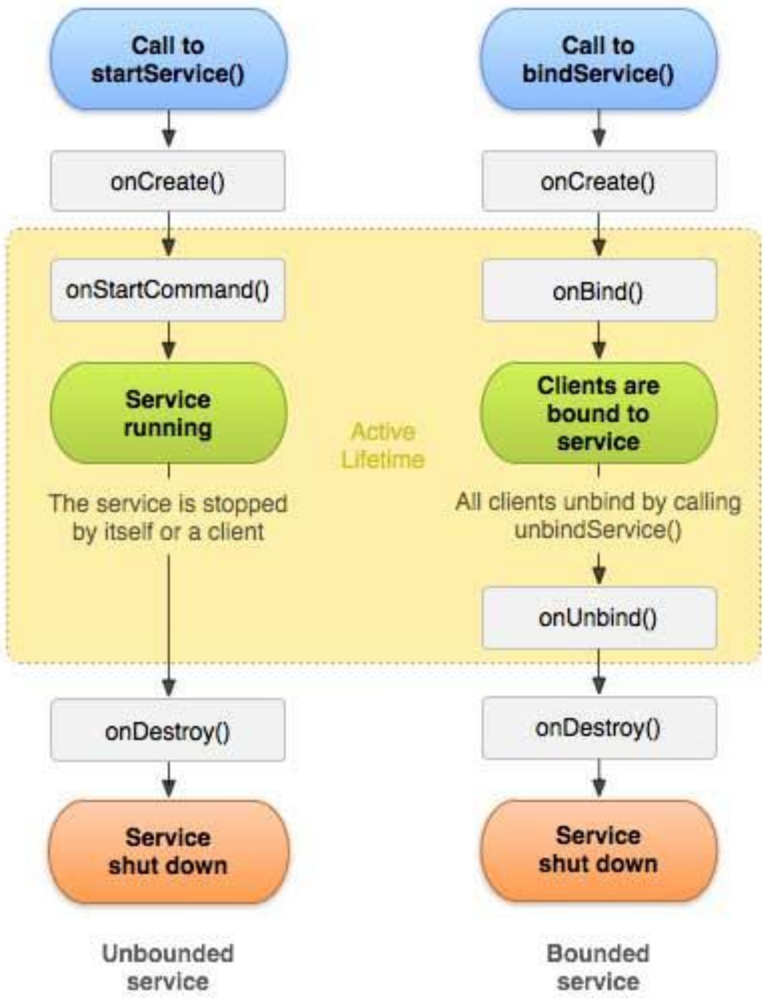
8

Services

A service is a component that runs in the background to perform long-running operations without needing to interact with the user. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. A service can essentially take two states:

State	Description
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.
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 lifecycle 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 lifecycle when the service is created with startService() and the diagram on the right shows the lifecycle 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.

Callback	**Description**
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.
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*.
onUnbind()	The system calls this method when all clients have disconnected from a particular interface published by the service.
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)*.

onCreate()	The system calls this method when the service is first created usingonStartCommand() or onBind(). This call is required to perform one-time setup.
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 lifecycle 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 Eclipse IDE 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* to add *startService()* and *stopService()* methods.
3	Create a new java file *MyService.java* under the package *com.example.helloworld*. 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 detault content of *res/layout/activity_main.xml* file to include two buttons in linear layout.
6	Define two constants *start_service* and *stop_service* in *res/values/strings.xml* file
7	Run the application to launch Android emulator and verify the result of the changes done in the aplication.

Following is the content of the modified main activity file**src/com.example.helloworld/MainActivity.java**. This file can include each of the fundamental lifecycle methods. We have added *startService()* and *stopService()* methods to start and stop the service.

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.content.Intent;
import android.view.View;

public class MainActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.activity_main, menu);
      return true;
   }

   // Method to start the service
   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 **src/com.example.helloworld/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.helloworld;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service {
   @Override
   public IBinder onBind(Intent arg0) {
      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:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.helloworld"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
   <application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/AppTheme" >
       <activity
           android:name=".MainActivity"
           android:label="@string/title_activity_main" >
           <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>
```

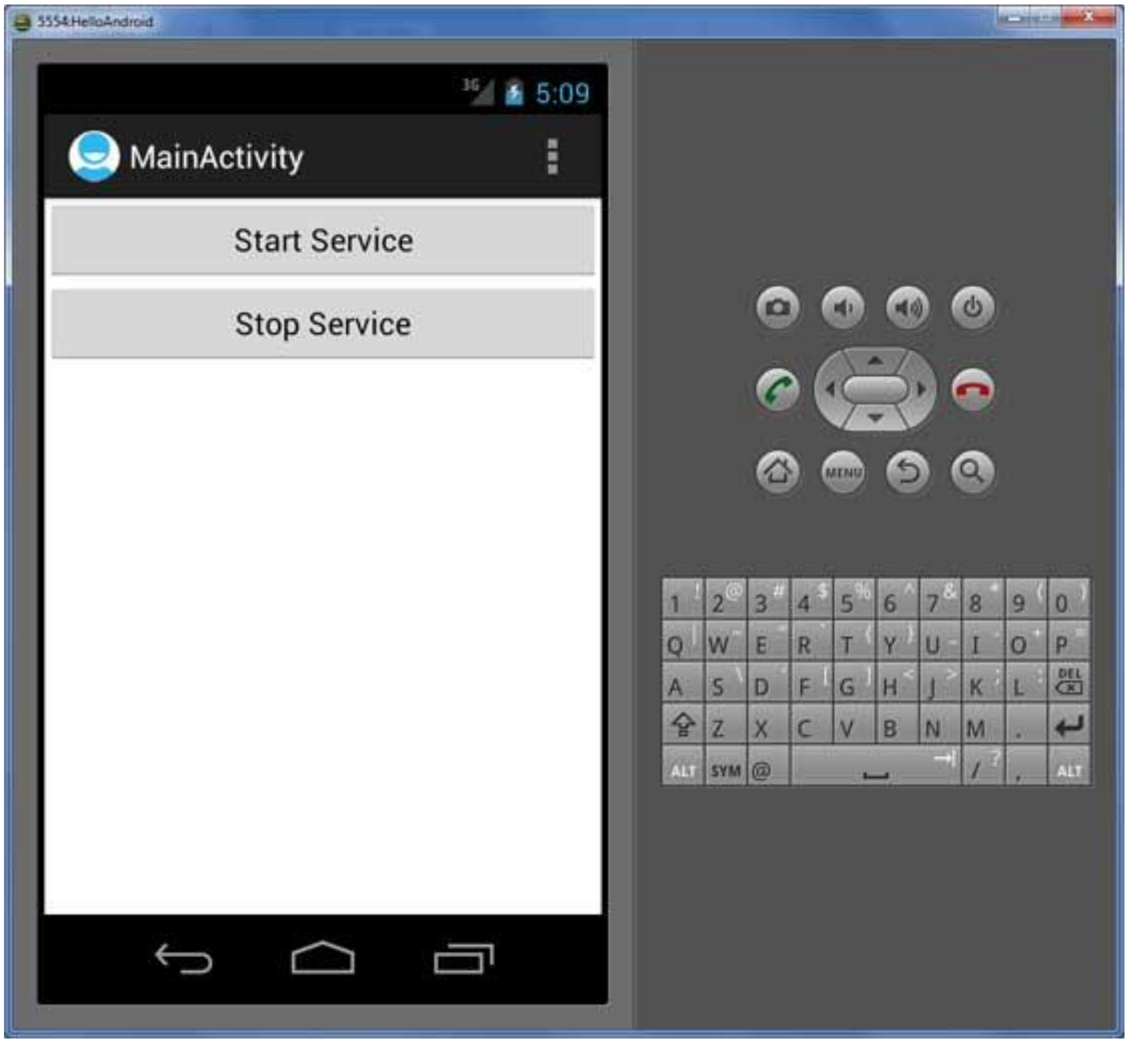
Following will be the content of **res/layout/activity_main.xml** file to include two buttons:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <Button android:id="@+id/btnStartService"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/start_service"
   android:onClick="startService"/>

   <Button android:id="@+id/btnStopService"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/stop_service"
   android:onClick="stopService" />

</LinearLayout>
```

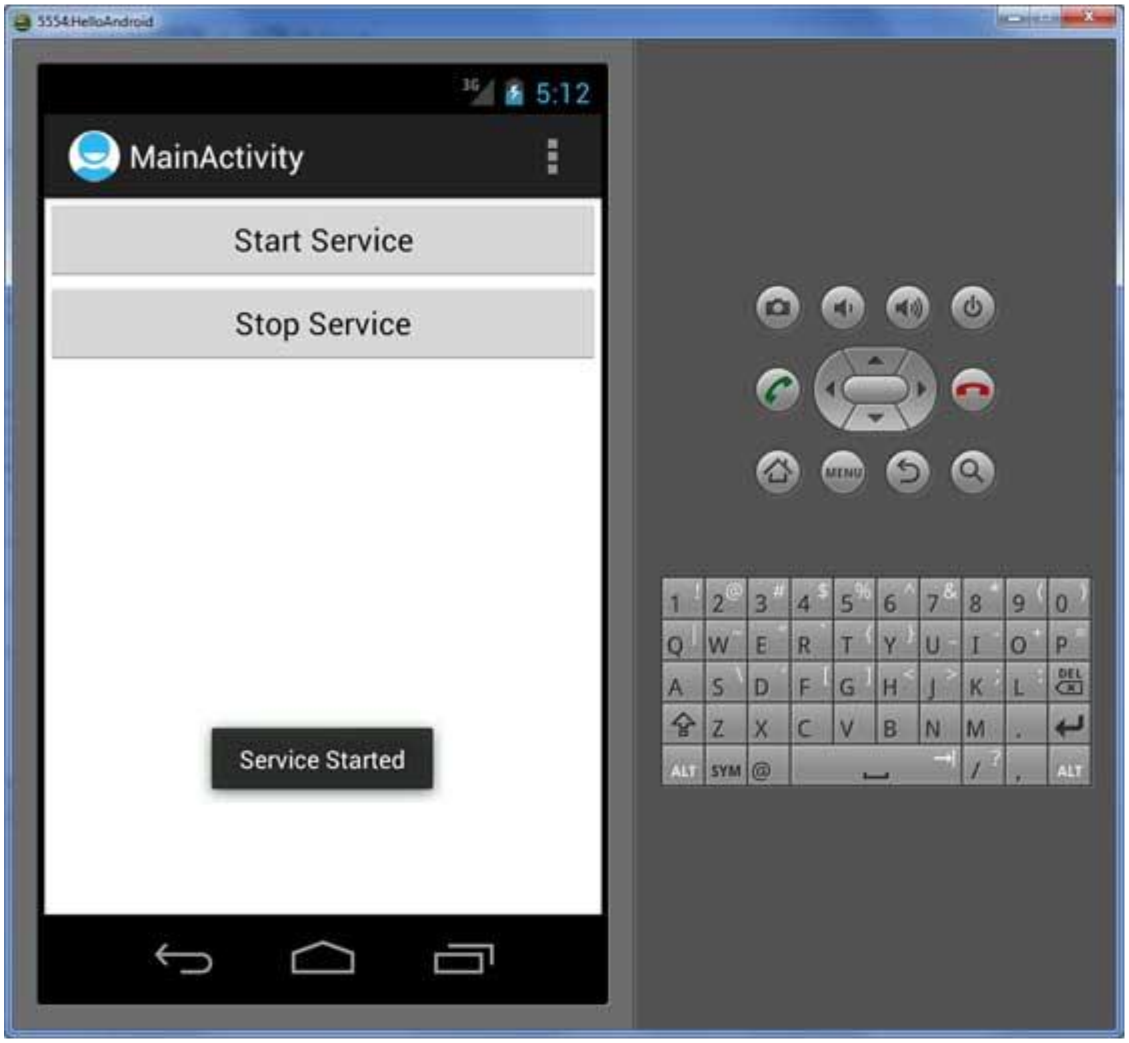
Following will be the content of **res/values/strings.xml** to define two new constants:

```
<resources>

    <string name="app_name">HelloWorld</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_main">MainActivity</string>
    <string name="start_service">Start Service</string>
    <string name="stop_service">Stop Service</string>

</resources>
```

Let's try to run our modified **Hello World!** application we just modified. I assume you had created your**AVD** while doing environment setup. To run the app from Eclipse, open one of your project's activity files and click Run icon from the toolbar. Eclipse installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window:



Now to start your service, let's click on **Start Service** button, this will start the service and as per our programming in *onStartCommand()* method, a message *Service Started* will appear on the bottom of the the simulator as follows:



To stop the service, you can click the Stop Service button.