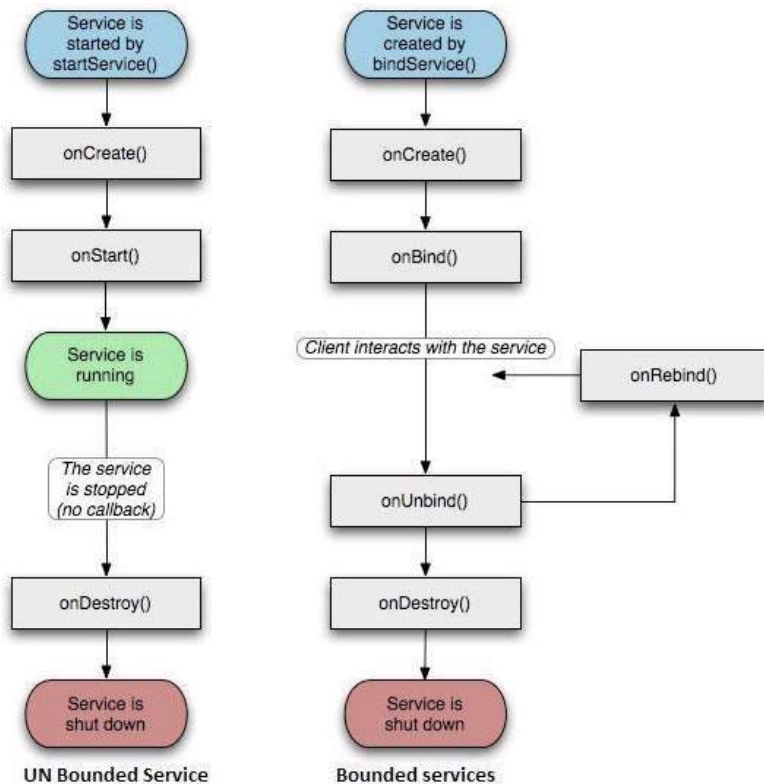


A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states – Started and Bound

State & Description
<p>Started</p> <p>A service is started when an application component, such as an activity, starts it by calling <code>startService()</code>. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.</p>
<p>Bound</p> <p>A service is bound when an application component binds to it by calling <code>bindService()</code>. 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 inter-process communication (IPC).</p>

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 a 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 callback methods.

Callback & Description
<p>onStartCommand()</p> <p>The system calls this method when another component, such as an activity, requests that the service be started, by calling <code>startService()</code>. If you implement this method, it is your responsibility to stop the service when its work is done, by calling <code>stopSelf()</code> or <code>stopService()</code> methods.</p>
<p>onBind()</p> <p>The system calls this method when another component wants to bind with the service by calling <code>bindService()</code>. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an <code>IBinder</code> object. You must always implement this method, but if you don't want to allow binding, then you should return <code>null</code>.</p>
<p>onUnbind()</p> <p>The system calls this method when all clients have disconnected from a particular interface published by the service.</p>
<p>onRebind()</p> <p>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 <code>onUnbind(Intent)</code>.</p>
<p>onCreate()</p> <p>The system calls this method when the service is first created using <code>onStartCommand()</code> or <code>onBind()</code>. This call is required to perform one-time set-up.</p>
<p>onDestroy()</p> <p>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.</p>

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

```
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() {

    }

}
```