Broadcast Receiver and Services

Broadcast Receiver

- A *broadcast receiver* is a component that responds to system-wide broadcast announcements.
- Broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.
- A broadcast receiver is implemented as a subclass of **BroadcastReceiver and each broadcast is delivered as an Intent object.**

Purpose of Broadcast Receiver

- Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
- Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

Steps to Use Broadcast Receiver

- Create Class of type Broadcast Receiver.
- Implement onReceive Method.
- Create and Configure Notification Manager.
- Define Pending Intents.
- Register Receiver in Manifest File.

Step:1 Create class of type Broadcast Receiver.

Step:2 Implement onReceive Method.

Step:3 Configure NotificationManager.

Step:4 Define Pending Intent

```
public class MyBroadcastReceiver extends BroadcastReceiver
{
    private NotificationManager mNotificationManager;
    private int SIMPLE_NOTFICATION_ID;

    @Override
    public void onReceive(Context context, Intent intent)
    {
        mNotificationManager = (NotificationManager)context.getSystemService(Context.NOTIFICATION_SERVICE);
        Notification notifyDetails = new Notification(R.drawable.android,"Time Reset!",System.currentTimeMillis());

        PendingIntent myIntent = PendingIntent.getActivity(context, 0, new Intent(Intent.ACTION_VIEW, People.CONTENT_URI), 0);
        notifyDetails.setLatestEventInfo(context, "Time has been Reset", "Click on me to view Contacts", myIntent);
        notifyDetails.flags |= Notification.FLAG_AUTO_CANCEL;
        notifyDetails.flags |= Notification.DEFAULT_SOUND;
        mNotificationManager.notify(SIMPLE_NOTFICATION_ID, notifyDetails);
        Log.i(getClass().getSimpleName(),"Sucessfully Changed Time");
    }
}
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.collabera.labs.sai"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
            <action android:name="android.intent.action.TIME_SET"/>

            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Register Broadcast Receiver.

Services

- A **Service is an application component that can perform long-running operations in the background and does not provide a user interface.**
- Another application component can start a service and it will continue to run in the background even if the user switches to another application.
- A component can bind to a service to interact with it and even perform interprocess communication (IPC).

A service can essentially take two forms:

- **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.**
- For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

- **Bound State:**

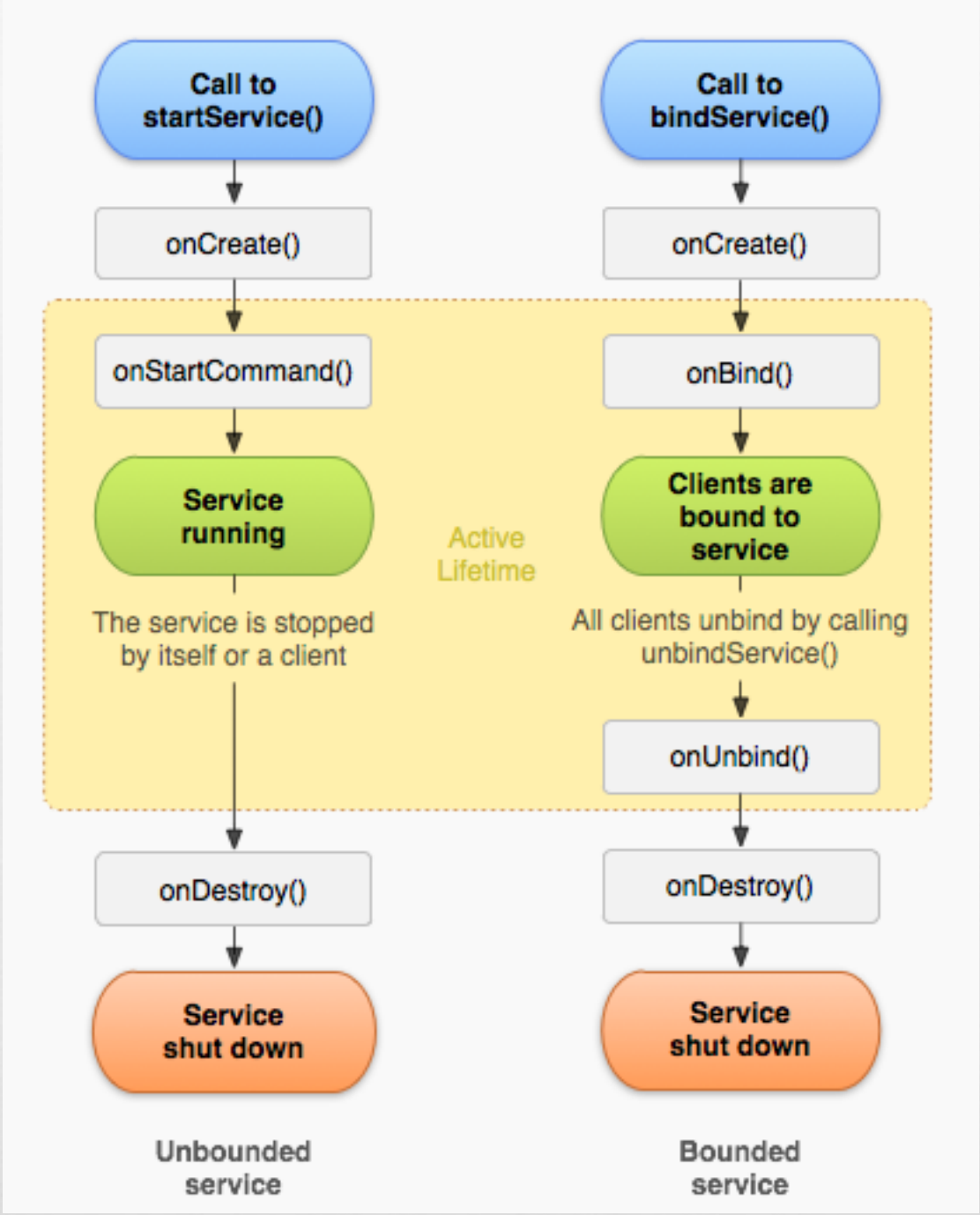
- A service is "bound" when an application component binds to it by calling **bindService().**

- **A bound service runs only as long as another application component is bound to it.**

- **Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.**

Basics terms used for Services

- **onStartCommand():**
- The system calls this method when another component, such as an activity, requests that the service be started, by calling **startService().**
- **Once this method executes, the service is started and can run in the background indefinitely.**
- **If you implement this, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService().**


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Active Lifetime
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service