FPT Software

ANDROID TRAINING LESSON 7

Version 0.1

Agenda

- Application Building Blocks
- Broadcast Receivers
- Pending Intent
- Services

Application Building Blocks

Activity	• UI Component Typically Corresponding to one screen.
IntentReceiver	• Responds to notifications or status changes. Can wake up your process.
Service	• Faceless task that runs in the background.
ContentProvider	• Enable applications to share data.

Android Application Anatomy

Activities

1. Provides User Interface
2. Usually represents a Single Screen
3. Can contain one/more Views
4. Extends the Activity Base class

Services

1. No User Interface
2. Runs in Background
3. Extends the Service Base Class

Application= Set of Android Components

Intent/Broadcast Receiver

1. Receives and Reacts to broadcast Intents
2. No UI but can start an Activity
3. Extends the BroadcastReceiver Base Class

Content Provider

1. Makes application data available to other apps
2. Data stored in SQLite database
3. Extends the ContentProvider Base class

Broadcast Receivers

1. A ***broadcast receiver*** is a component that responds to system-wide Broadcast announcements.
2. A broadcast receiver is registered as a receiver in an Android Application via the AndroidManifest.xml file, or You can also register BroadcastReceiver dynamically via the Context.registerReceiver() method.
3. Many broadcasts originate from the Android system—for example, a Broadcast announcing that the screen has turned off, the battery is low, or a picture was captured or an SMS is received.

Broadcast Receivers

- A ***broadcast receiver*** will be able to receive intents which can be generated via the Context.sendBroadcast() method.
- The class BroadcastReceiver defines the onReceive() method. Only during this method your BroadcastReceiver object will be valid, afterwards the Android system can recycle the BroadcastReceiver. Therefore you cannot perform any asynchronous operation in the onReceive() method. .

Pending Intent

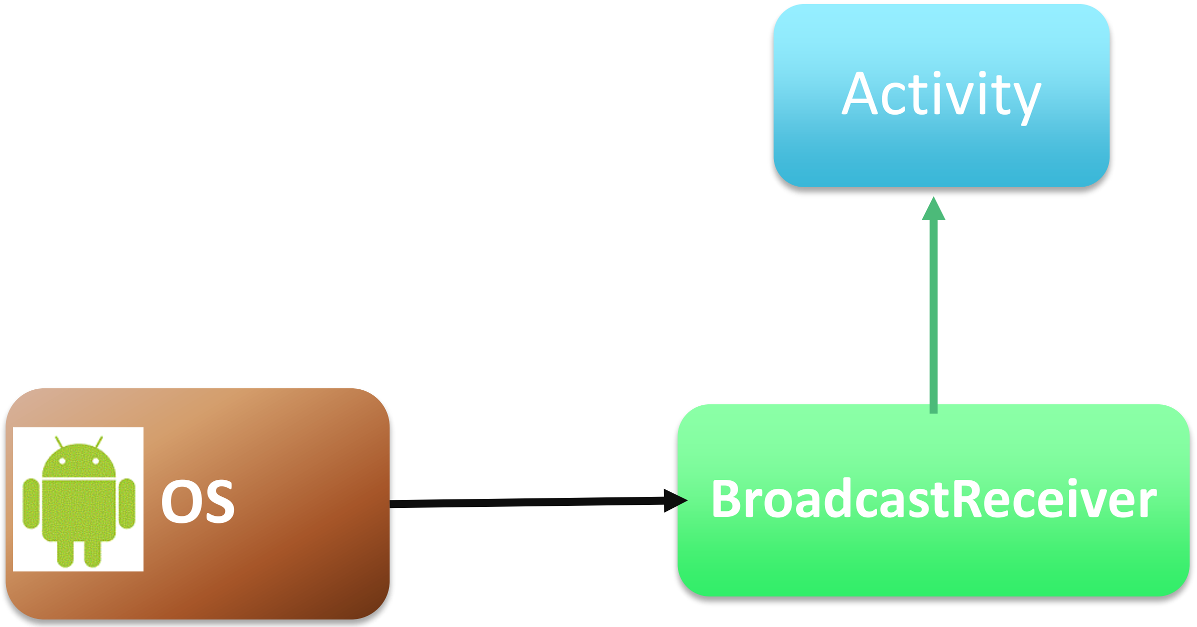
- A PendingIntent is a token that you give to another application (e.g. Notification Manager, Alarm Manager or other 3rd party applications), which allows this other application to use the permissions of your application to execute a predefined piece of code.
- To perform a broadcast via a pending intent so get a PendingIntent via PendingIntent.getBroadcast().
- To perform an activity via an pending intent you receive the activity via PendingIntent.getActivity().

Pending Intent

- There are three static PendingIntent*factory* methods which can be used to obtain a PendingIntent
  - public static PendingIntent getActivity(Context context, int requestCode, Intent intent, int flags)
  - public static PendingIntent getBroadcast(Context context, int requestCode, Intent intent, int flags)
  - public static PendingIntent getService(Context context, int requestCode, Intent intent, int flags)
- These return PendingIntent instances which can be used to
  - start an activity,
  - perform a broadcast, or
  - start a service
- Depending upon the given arguments each of these methods can either
  - create a new PendingIntent object,
  - modify an existing PendingIntent object,
  - modify an existing PendingIntent object and create a new PendingIntent object, or
  - do nothing

Broadcast Receivers

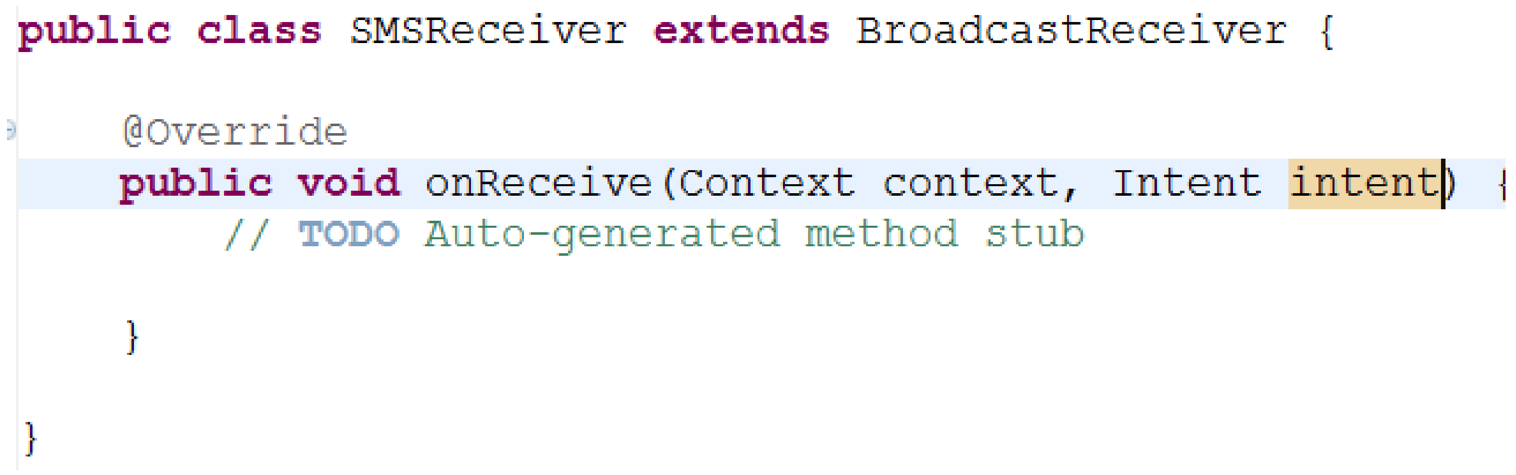
1. We'll use a Broadcast Receiver to capture SMS receive event
2. We capture the SMS receive event and launch an Activity to show the sms and give user an option to reply the SMS



Broadcast Receivers

1. Create a new project **BroadcastReceiverDemo**
2. A broadcast receiver is implemented as a subclass of **BroadcastReceiver** and each broadcast is delivered as an **Intent** object. In this case the intent is detected by android.provider.Telephony.SMS_RECEIVED

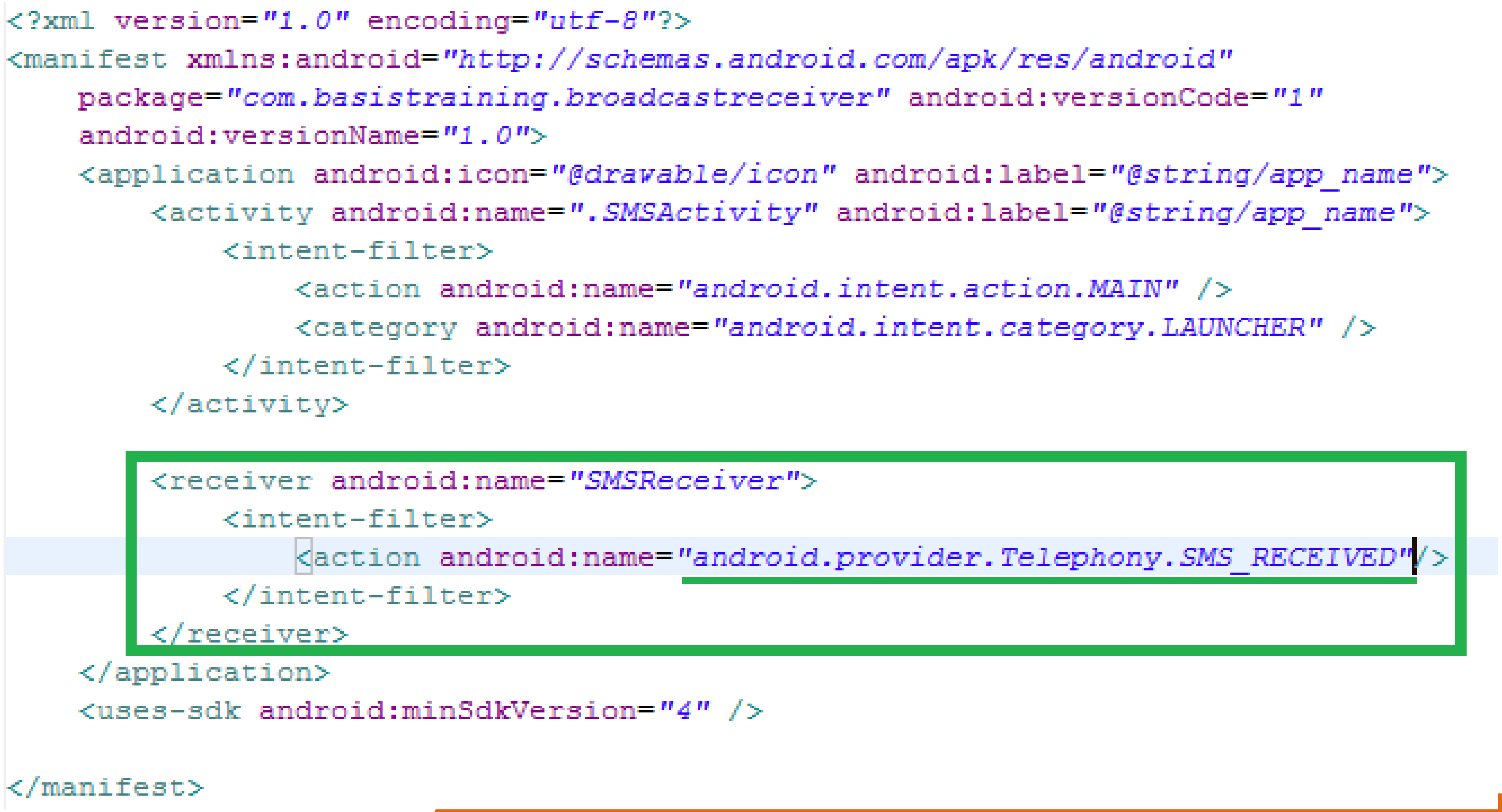
   To do this we'll create a class **SMSReceiver** that extends **BroadcastReceiver** class and define the method **onReceive()**



```
public class SMSReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO Auto-generated method stub

    }

}
```

Broadcast Receivers

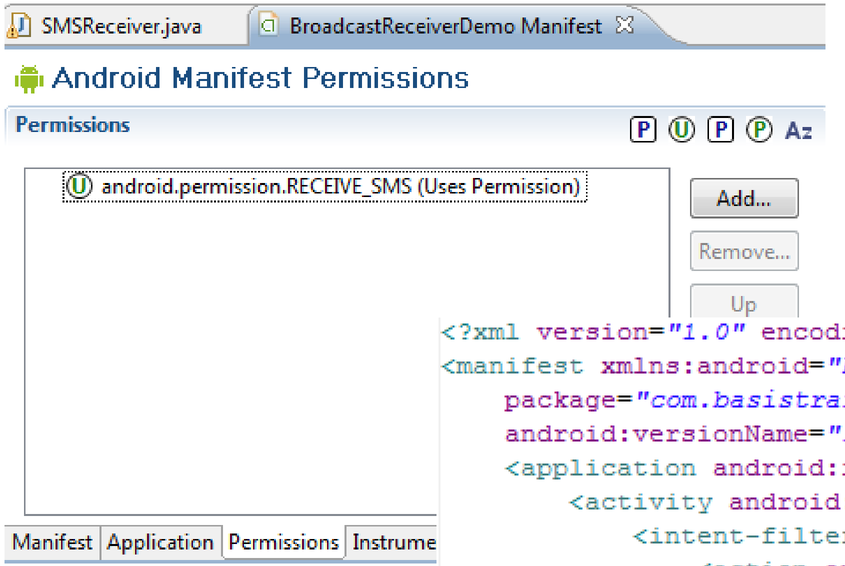
3. We also need to add **SMSReceiver** as receiver of a particular Intent (SMS received) which is identified by *android.provider.Telephony.SMS_RECEIVED*



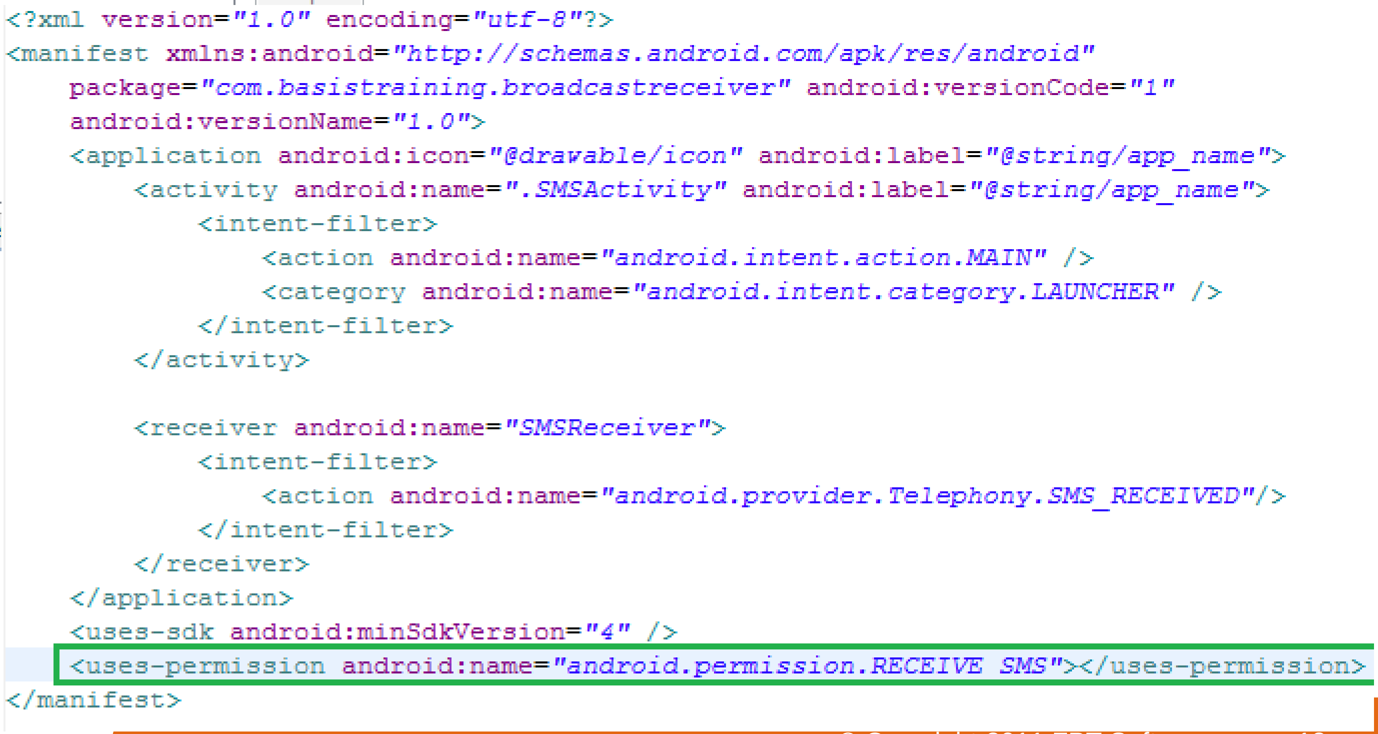
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.basistraining.broadcastreceiver" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SMSActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name="SMSReceiver">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="4" />

</manifest>
```

Broadcast Receivers

4. Also we have to add permission for receiving SMS



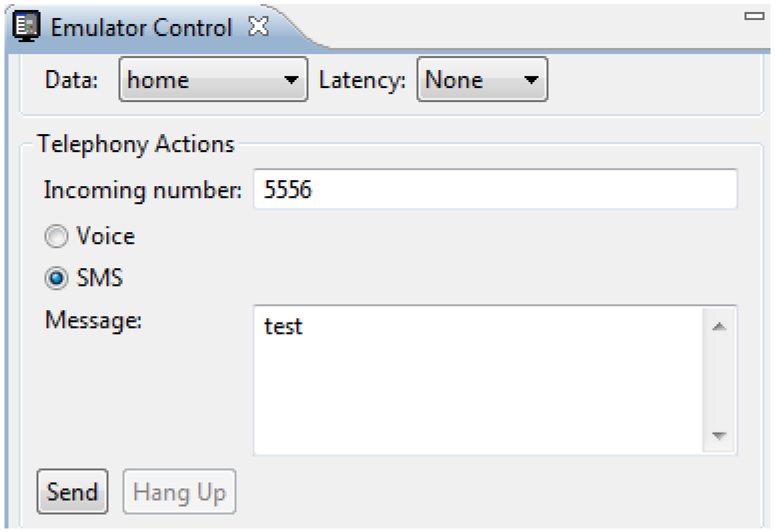
android.permission.RECEIVE_SMS (Uses Permission)



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.basistraining.broadcastreceiver" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SMSActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name="SMSReceiver">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="4" />
    <uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
</manifest>
```

Broadcast Receivers

5. Now we run the application
6. Now we use emulator control to send sms



Emulator Control

Data: home Latency: None

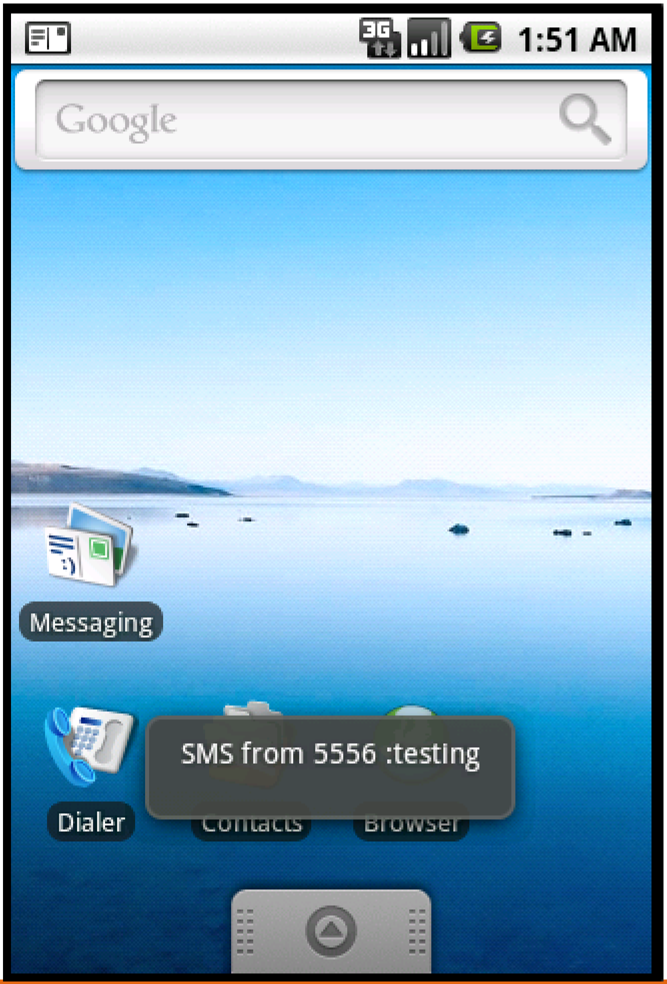
Telephony Actions

Incoming number: 5556

Voice

[✓ SMS]

Message: test



SMS from 5556 :testing

Receiving SMS

```
Bundle bundle = intent.getExtras();
SmsMessage[] msgs = null;
String str = "";
String address="";
if (bundle != null)
{
//---retrieve the SMS message received---
Object[] pdus = (Object[]) bundle.get("pdus");
msgs = new SmsMessage[pdus.length];
for (int i=0; i<msgs.length; i++)
            {
              msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
              str += "SMS from " + msgs[i].getOriginatingAddress();
              str += " :";
              str += msgs[i].getMessageBody().toString();
              str += "\n";
              address=msgs[i].getOriginatingAddress();
              Toast.makeText(context, str, Toast.LENGTH_LONG).show();

            }
```

Sending SMS

1. Add permission in menifest.xml

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

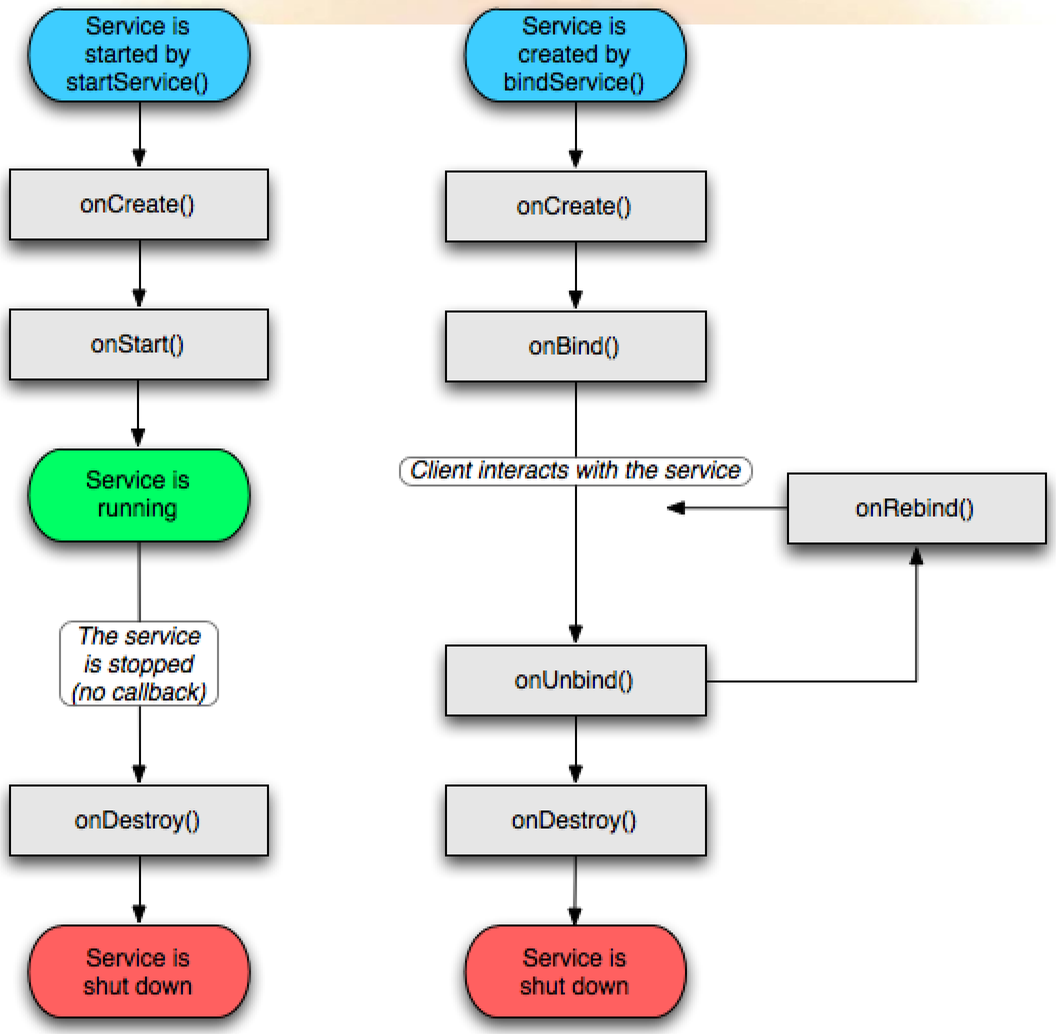
2. We add the following code for sending SMS from anywhere of our application

```
SmsManager sm = SmsManager.getDefault();
String number = "5556";
sm.sendTextMessage(number, null, "Test SMS Message", null, null);
```

Services

- **runs in the same process as the application it is part of.**
- **To create a application to run in the background of other current activities, one needs to create a Service.** The Service can run indefinitely (unbounded) or can run at the lifespan of the calling activity(bounded).

Services


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Services

- a Service has a different lifecycle than activities therefore have different methods. But to begin a service in the application a call to **startService()** which envokes the service **onCreate()** method and **onStart()** beginning running the service.

  context.startService() | ->onCreate() - >onStartCommand() [service running]

  Calling the applications **stopService()** method to stop the service.

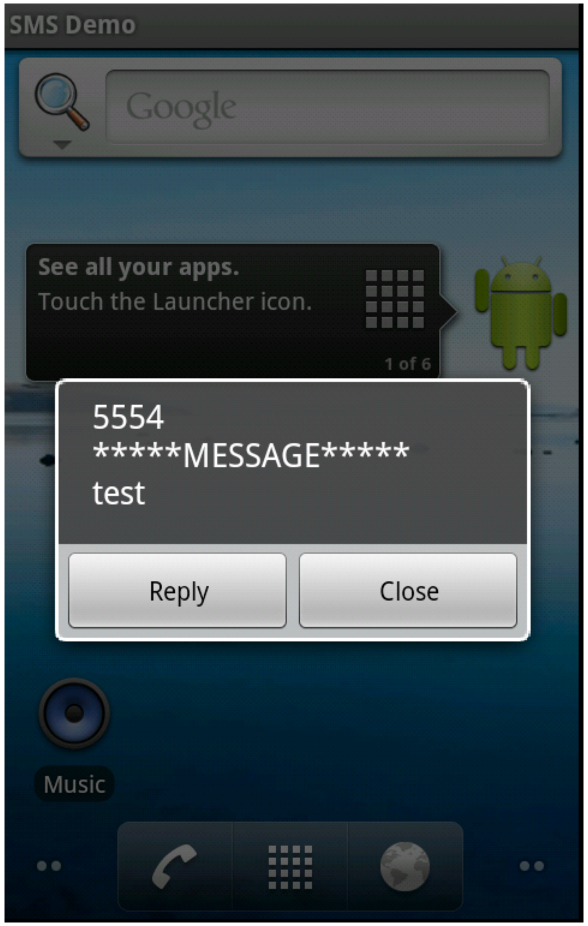
  context.stopService() | ->onDestroy() [service stops]

  context.onBindService() | ->onCreate() [service created] . The client will receive the IBinder object that the service returns from its onBind(Intent) method, allowing the client to then make calls back to the service.

Homework

- Create SMS application:
  - Display incoming phone number
  - Quick reply
  - close

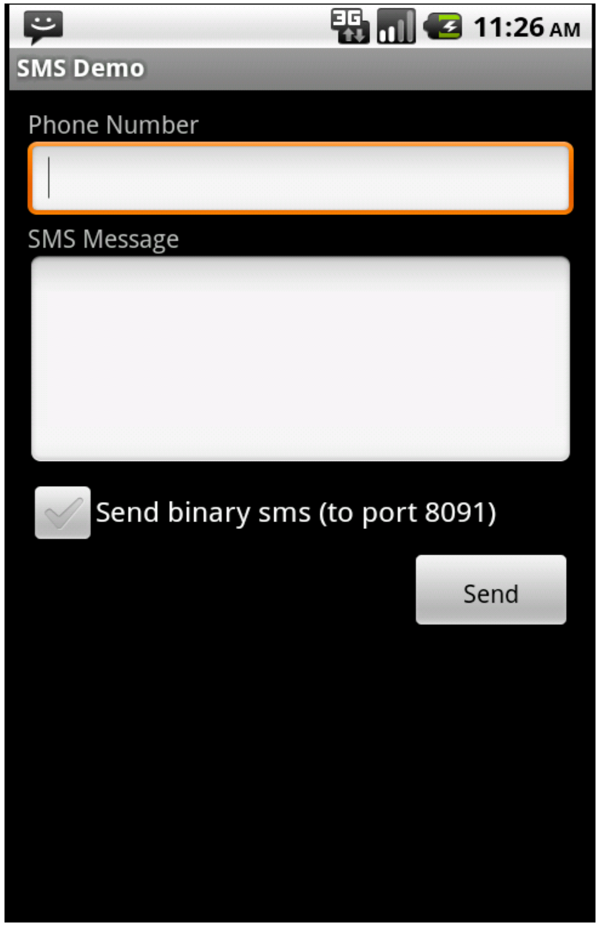
Homework


SMS Demo

See all your apps.
Touch the Launcher icon.

1 of 6

5554
*****MESSAGE*****
test


11:26 AM

SMS Demo

Phone Number

SMS Message

Send binary sms (to port 8091)

Thank you!