Ubiquitous Computing

COMP H4025

Lecturer: Simon McLoughlin

Lecture 7

Lecture Overview

This week:

- Services
- Broadcast Receivers/Pending Intents
- Notifications

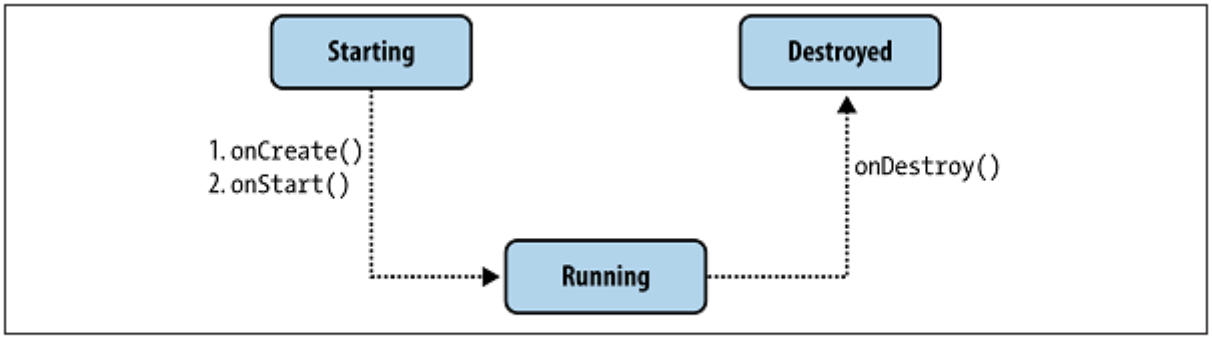
Services

- Services are among the main building blocks in Android. Unlike an activity, a service doesn't have a user interface; it is simply a piece of code that runs in the background of your application.
- Services are used for processes that should run independently of activities, which may come and go. A twitter like application, for example, may need to create a service to periodically connect to the cloud and check for new statuses from the user's friends.
- Example: Google Play Music plays the music using a service.
- Example: Web browser runs a downloader service to retrieve a file
- The steps in creating a Service are:

    1. Create the Java class representing your service.
    2. Register the service in the AndroidManifest.xml file.
    3. Start the service.

Creating a Service

- The basic procedure for creating a service, as with activities and other main building blocks, is to subclass a Service class provided by the Android framework.

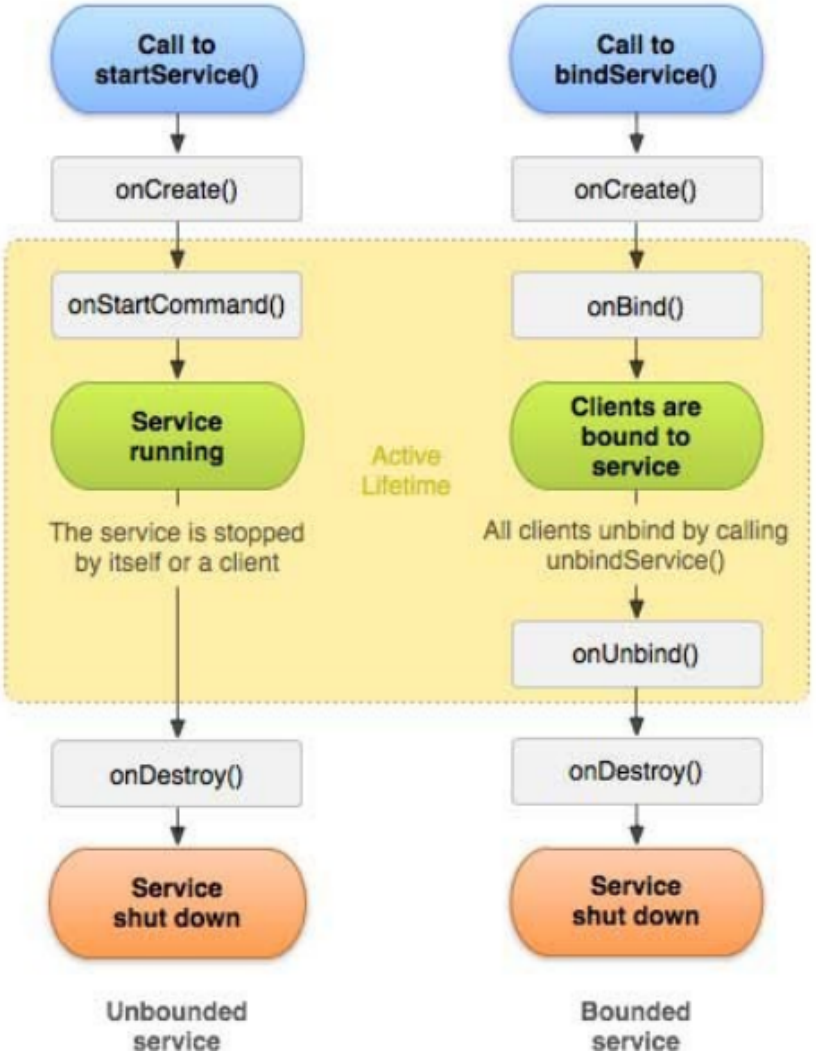


Service life cycle

- Next, we want to override some of the main life cycle methods, onCreate() is called when the service is created for the first time.
- onStartCommand() is called each time the service is started
- onDestroy() is called when the service is terminated
- A service can be bound or unbound. We'll focus on unbound services, where the life cycle of a service is not tied to the life cycle of the activities that started them.

Service Life Cycle

- A service is started by an app's activity using an intent.
- Service operation modes:
  - **start**: The service keeps running until it is manually stopped.
    - *we'll use this one*
  - **bind**: The service keeps running until no "bound" apps are left.
- Services have similar methods to activities for lifecycle events.
  - `onCreate`, `onDestroy`

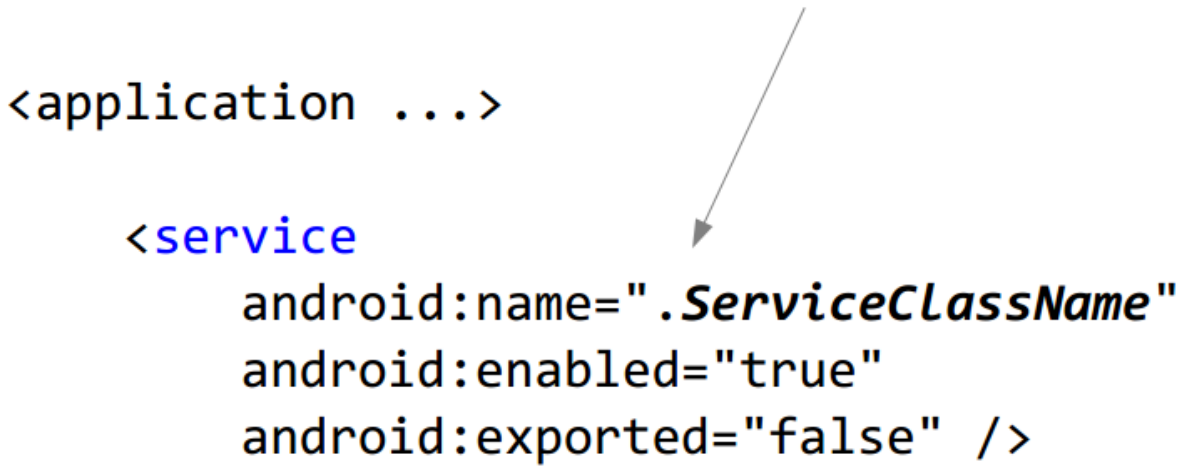


Manifest Entry

- To allow your app to use the service, add the following to your app's `AndroidManifest.xml` configuration:

  (Android Studio does this for you if you use the New Service option)

  - the `exported` attribute signifies whether other apps are also allowed to use the service (true=yes, false=no)
  - note that you must write a dot (`.`) before the class name below!



```
<application ...>

    <service
        android:name=".ServiceClassName"
        android:enabled="true"
        android:exported="false" />
```

Service Template

```
public class ServiceClassName extends Service {
    /* this method handles a single incoming request */
    @Override
    public int onStartCommand(Intent intent, int flags, int id) {
        // unpack any parameters that were passed to us
        String value1 = intent.getStringExtra("key1");
        String value2 = intent.getStringExtra("key2");

        // do the work that the service needs to do ...

        return START_STICKY;   // stay running
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;   // disable binding
    }
}
```

RefreshService class

```java
package com.marakana.android.yamba;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class RefreshService extends Service {
  static final String TAG = "RefreshService"; // ❶

  @Override
  public IBinder onBind(Intent intent) { // ❷
    return null;
  }

  @Override
  public void onCreate() { // ❸
    super.onCreate();
    Log.d(TAG, "onCreated");
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) { // ❹
    super.onStartCommand(intent, flags, startId);
    Log.d(TAG, "onStarted");
    return START_STICKY;
  }

  @Override
  public void onDestroy() { // ❺
    super.onDestroy();
    Log.d(TAG, "onDestroyed");
  }
}
```

RefreshService

- To keep it simple this service currently just adds something to the log. It will eventually do something more meaningful.

- 1. As in all major classes, we like to add the TAG constant because we use Log.d() quite a bit.

- 2. onBind() is used in bound services to return the actual implementation of something called a binder. Because we are not using a bound service, we can just return null here.

- 3. onCreate() is called when the service is initially created. It is not called for subsequent startService() calls, so it is a good place to do work that needs to be done only once during the life of a service.

- 4. onStartCommand() is called each time the service receives a startService()intent. A service that is already started could get multiple requests to start again, and each will cause onStartCommand() to execute. START_STICKY is used as a flag to indicate this service will be restarted should it be killed by the system before it finishes its job.

- 5. onDestroy() is called just before the service is destroyed by the stopService() request. This is a good place to clean up things that might have been initialized in onCreate().

Starting a Service

- In your `Activity` class:

```
Intent intent = new Intent(this, ServiceClassName.class);
intent.putExtra("key1", "value1");
intent.putExtra("key2", "value2");
startService(intent);   // not startActivity!
```

- or if the same code is launched from a fragment:

```
Intent intent = new Intent(getActivity(),
                           ServiceClassName.class);
```

Finishing a Service

- When a service has completed a task, it can notify the app by "sending a broadcast" which the app can listen for:
  - As before, set an **action** in the intent to distinguish different kinds of results.

```
public class ServiceClassName extends Service {
    @Override
    public int onStartCommand(Intent tent, int flags, int id) {
        // do the work that the service needs to do ...
        ...
        // broadcast that the work is done
        Intent done = new Intent();
        done.setAction("action");
        done.putExtra("key1", value1); ...
        sendBroadcast(done);

        return START_STICKY;   // stay running
    }
```

Receiving a Broadcast

- Your activity can hear broadcasts using a BroadcastReceiver.
  - Extend BroadcastReceiver with the code to handle the message.
  - Any extra parameters in the message come from the service's intent.

```
public class ActivityClassName extends Activity {
    ...

    private class ReceiverClassName extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            // handle the received broadcast message
            ...
        }
    }
```

Receiving a Broadcast

- Set up your activity to be notified when certain broadcast actions occur.
  - You must pass an **intent filter** specifying the action(s) of interest.

```
public class ActivityClassName extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        IntentFilter filter = new IntentFilter();
        filter.addAction("action");
        registerReceiver(new ReceiverClassName(), filter);
    }
```

More on Broadcast Receivers later…….

IntentService

- It’s important to note that your service is going to run on the main thread of the application, i.e., the UI thread.
- Because our service is going to be connecting to the cloud to pull down the latest data, we once again have the problem of networking on the UI thread.
- Android SDK provides a subclass of service called IntentService that is similar to a regular service, with two main exceptions: whatever work has to be done in onHandleIntent() will execute on a separate worker thread, and once it’s done, the service will stop.

RefreshService as an IntentService

```
package com.marakana.android.yamba;

import android.app.IntentService;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

public class RefreshService extends IntentService {
    static final String TAG = "RefreshService"; // ❶

    public RefreshService() { // ❷
                super(TAG);
        }


    @Override
    public void onCreate() { // ❸
        super.onCreate();
        Log.d(TAG, "onCreated");
    }

    // Executes on a worker thread
        @Override
        protected void onHandleIntent(Intent intent) { // ❹
        Log.d(TAG, "onStarted");
    }

    @Override
    public void onDestroy() { // ❺
        super.onDestroy();
        Log.d(TAG, "onDestroyed");
    }
}
```

RefreshService as an IntentService

- 1. This is the usual tag that we'll use for logging.
- 2. IntentService requires a default constructor. In that constructor, you need to call super() and pass a name of this service. TAG variable comes in handy for this.
- 3. Just as in a regular service, onCreate() is called when the service is created for the first time.
- 4. onHandleIntent() is where we do the main work. This work will be executed on a separate thread. This is one of the main differences between a service and an intent service.
- 5. Just as in a regular service, onDestroy() is called when the service is about to be stopped. Unlike a regular service, onDestroy() is called as soon as onHandleIntent() terminates.
- Services also have to be added to the application part of the manifest file:

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
    ...
    <service android:name=".RefreshService" /> <!-- ❶ -->
    ...
</application>
```

Broadcast Receivers

• A broadcast receiver is an Android component which allows you to **register for system or application events**. All registered receivers for an event are notified by the Android runtime once this event happens.

• For example, applications can register for the **ACTION_BATTERY_LOW** system event which is fired once the Android system detects the battery has dropped beyond a certain level

• A receiver can be **registered via the AndroidManifest.xml** file or registered dynamically via the Context.registerReceiver() method.

•The implementing class for a receiver **extends the BroadcastReceiver** class and if the event for which the broadcast receiver has registered happens, the **onReceive() method** of the receiver is called by the Android system.

• The Android system excludes all receivers from receiving intents by default if the corresponding application has **never been started** by the user or if the user **explicitly stopped** the application via the Android menu.

Broadcast Receiver for Phone State

• Here is an example that defines a broadcast receiver which listens to telephone state changes. If the phone receives a phone call, then the receiver will be notified and log a message.

```
<uses-sdk android:minSdkVersion="15" />

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

<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
       <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>

    <receiver android:name="MyPhoneReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.PHONE_STATE" >
            </action>
        </intent-filter>
    </receiver>
</application>
```

Broadcast Receiver for Phone State

• Here is the receiver class that will be launched in reaction to a phone call, note in this example a Bundle object is just another way to pass data between android components.

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.util.Log;

public class MyPhoneReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extras = intent.getExtras();
    if (extras != null) {
      String state = extras.getString(TelephonyManager.EXTRA_STATE);
      Log.w("MY_DEBUG_TAG", state);
      if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
        String phoneNumber = extras
            .getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
        Log.w("MY_DEBUG_TAG", phoneNumber);
      }
    }
  }
}
```

Pending Intents

• If you want some action to take place at **some point in the future** in response to some event you should use a PendingIntent instead of an Intent object.

• For example, lets say you want to set an alert or an alarm in your activity, when the alarm goes off you want an Intent to be started and a Broadcast Receiver to then react to this Intent. As you do not want the Intent to be started straight away but rather when the Alarm goes off you should use a PendingIntent.

• When you start or pass a PendingIntent to another Android component, that component performs the action described by the Intent with the **same set of permissions** as the component that created the Intent.

• In the example of an Activity that wants to invoke some action using an Intent in response to some event, the activity **might not be around** to launch the Intent when the event occurs so it should specify a Pending Intent, so even if the Activity process is stopped the component can still be launched by the Pending Intent.

BroadcastReceiver and Pending Intents

• Here is an example to schedule a receiver via the Android alarm manager system service. The receiver will vibrate the phone and display a toast when it is called.

• We must register the receiver in manifest as before and give the app permission to vibrate the device

```
<uses-sdk android:minSdkVersion="15" />

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

<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
    <activity
        android:name=".AlarmActivity"
        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="MyBroadcastReceiver" >
    </receiver>
</application>
```

BroadcastReceiver class

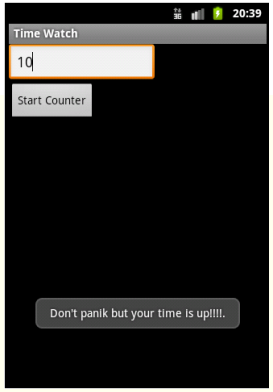
• Here is the broadcast receiver class that displays the Toast and vibrates the phone

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.widget.Toast;

public class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Don't panik but your time is up!!!!.",
        Toast.LENGTH_LONG).show();
    // Vibrate the mobile phone
    Vibrator vibrator = (Vibrator) context.getSystemService
(Context.VIBRATOR_SERVICE);
    vibrator.vibrate(2000);
  }

}
```

Activity to Set Alarm and define PendingIntent




```
import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class AlarmActivity extends Activity {

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

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void startAlert(View view) {
    EditText text = (EditText) findViewById(R.id.time);
    int i = Integer.parseInt(text.getText().toString());
    Intent intent = new Intent(this, MyBroadcastReceiver.class);
    PendingIntent pendingIntent = PendingIntent.getBroadcast
(this.getApplicationContext(), 234324243, intent, 0);
    AlarmManager alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    alarmManager.set(AlarmManager.RTC_WAKEUP, System.currentTimeMillis()
        + (i * 1000), pendingIntent);
    Toast.makeText(this, "Alarm set in " + i + " seconds",
        Toast.LENGTH_LONG).show();
  }

}
```

```
// Set the alarm to start at approximately 2:00 p.m.
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.set(Calendar.HOUR_OF_DAY, 14);

// With setInexactRepeating(), you have to use one of the AlarmManager interval
// constants--in this case, AlarmManager.INTERVAL_DAY.
alarmMgr.setInexactRepeating(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(),
        AlarmManager.INTERVAL_DAY, alarmIntent);
```

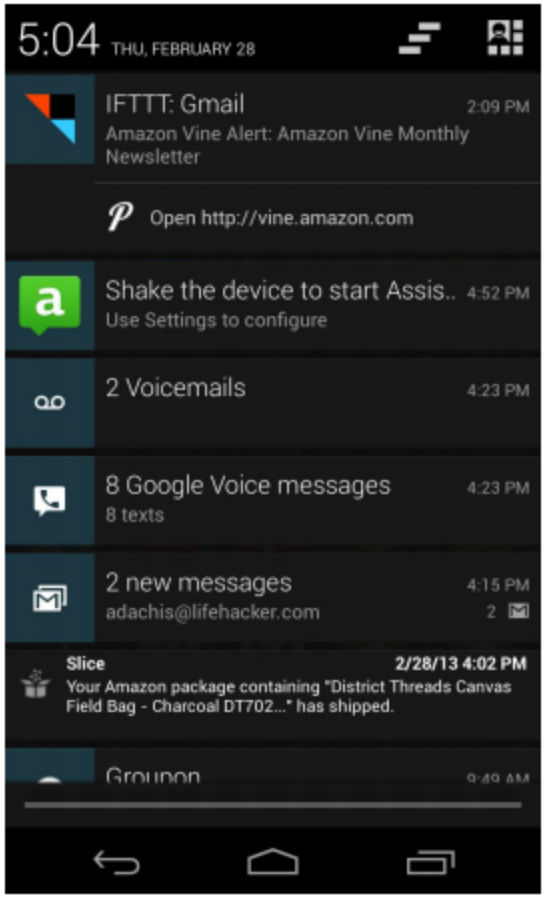
- Note here that the PendingIntent is going to perform a Broadcast as can be seen by PendingIntent.getBroadcast(..) but other options such as PendingIntent.getActivity() are also available to use a pending intent to start a new activity.
- Note also that a PendingInent requires an Intent object but not vice versa

Notifications

• A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the **notification area**.

• To see the details of the notification, the user opens the **notification drawer**. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

• A Notification object must contain the following:
  -A small icon, set by setSmallIcon()
  -A title, set by setContentTitle()
  -Detail text, set by setContentText()



Creating Notifications in Android

- Create a notification using a `NotificationBuilder`.
- Use `NotificationManager` to send out the notification.

```
Notification.Builder builder = new Notification.Builder(this)
        .setContentTitle("title")
        .setContentText("text")
        .setAutoCancel(true)
        .setSmallIcon(R.drawable.icon);
Notification notification = builder.build();

NotificationManager manager = (NotificationManager)
        getSystemService(Context.NOTIFICATION_SERVICE);
manager.notify(ID, notification);
```

• The Notification.Builder provides a builder interface to create an Notification object. You use a PendingIntent to specify the action which should be performed once the user selects the notification.

•The Notification.Builder allows you to add up to three buttons with definable actions to the notification.

Creating Notifications in Android

```
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class CreateNotificationActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void createNotification(View view) {
    // Prepare intent which is triggered if the
    // notification is selected
    Intent intent = new Intent(this, NotificationReceiverActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent, 0);

    // Build notification
    // Actions are just fake
    Notification noti = new Notification.Builder(this)
        .setContentTitle("New mail from " + "test@gmail.com")
        .setContentText("Subject").setSmallIcon(R.drawable.icon)
        .setContentIntent(pIntent)
        .addAction(R.drawable.icon, "Call", pIntent)
        .addAction(R.drawable.icon, "More", pIntent)
        .addAction(R.drawable.icon, "And more", pIntent).build();
    NotificationManager notificationManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
    // hide the notification after its selected
    noti.flags |= Notification.FLAG_AUTO_CANCEL;

    notificationManager.notify(0, noti);

  }
}
```

Creating Notifications in Android

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:onClick="createNotification"
        android:text="Create Notification" >
    </Button>

</LinearLayout>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is the result activity opened from the notification" >
    </TextView>

</LinearLayout>
```

```java
import android.app.Activity;
import android.os.Bundle;

public class NotificationReceiverActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.result);
  }
}
```