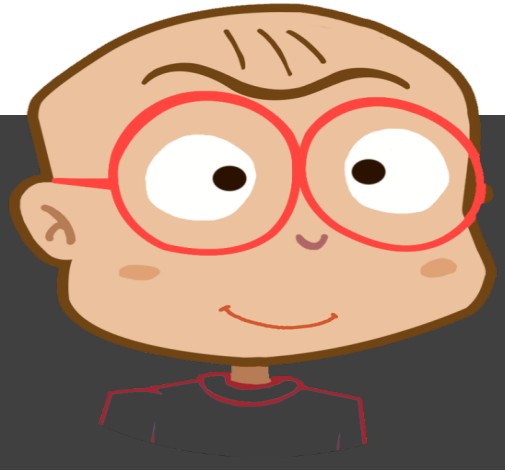


Intent and Broadcast Receiver

START »

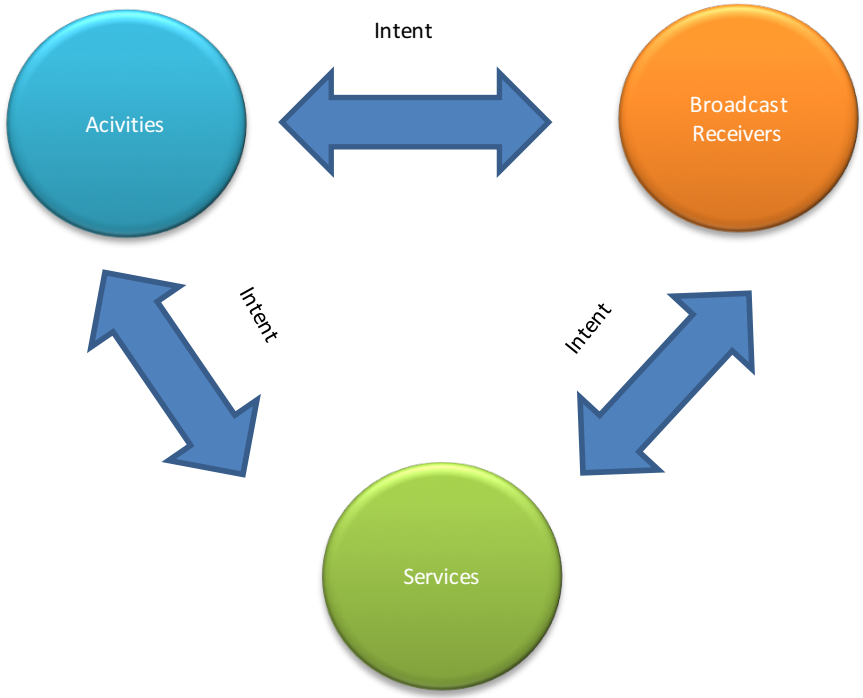
Learning Goals

After the course, attendees will be able to:

- Understanding about Intent and implement of Intent in Android
- Understanding about Broadcast Receiver and implement of Broadcast Receiver in Android

What is Intent?

Intents are **asynchronous messages** which allow Android **components** to request functionality from other **components** of the Android system.



Intent structure

Intent Object Structure Is Made Of

- Component name
- Action
- Data
- Category
- Extras
- Flags

Intent structure - Component name

Component name Field

- Specifies the name of the component (name of the activity if the component is activity) that should handle the intent

  >Class name of the target component (for example "com.javapassion.ForwardTargetActivity")

- Setting component name is optional

  If it is set, the Intent object is delivered to an instance of the designated class.
  If it is not set, Android uses other information in the Intent object to locate a suitable target

Intent structure - Action

Action Field

- A string naming the action to be performed
- The Intent class defines a number of pre-defined action constants, including ACTION_CALL, ACTION_EDIT, ACTION_MAIN, ACTION_SYNC, ACTION_BATTERY_LOW, etc.
- You can also define your own action strings for activating the components in your application
- The action largely determines how the rest of the intent is structured - particularly the data and extras fields - much as a method name determines a set of arguments and a return value.

Intent structure - Data

Data Field

- The URI of the data to be acted on and the MIME type of that data.
- Different actions are paired with different kinds of data specifications.

  If the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing.
  If the action is ACTION_CALL, the data field would be a tel: URI with the number to call.
  If the action is ACTION_VIEW and the data field is an http: URI, the receiving activity would be called upon to download and display whatever data the URI refers to.

Intent structure - Category

Category Field

- A string containing additional information about the kind of component (activity, service, or broadcast receiver) that should handle the intent.
- Any number of category descriptions can be placed in an Intent object
- Android provides a set of predefined categories (We will see them in the following slide)
- You can define your own categories

Intent structure - Category

Some Android Standard Categories

- CATEGORY_BROWSABLE - The target activity can be invoked within the browser to display data referenced by a link — for example, an image or an e-mail message.
- CATEGORY_GADGET - The activity can be embedded inside of another activity that hosts gadgets
- CATEGORY_HOME - This is the home activity, that is the first activity that is displayed when the device boots.
- CATEGORY_LAUNCHER - The activity can be the initial activity of a task and is listed in the top-level application launcher.
- CATEGORY_PREFERENCE - The target activity is a preference panel.
- Many more

Intent structure – Extras

Extras Field

- Key-value pairs for additional information that should be delivered to the component handling the intent.
- Just as some actions are paired with particular kinds of data URIs, some are paired with particular extras.

  ACTION_TIMEZONE_CHANGED action has a "time-zone" extra that identifies the new time zone

  ACTION_HEADSET_PLUG action has a "state" extra indicating whether the headset is now plugged in or unplugged , as well as a "name" extra for the type of headset

  If you were to invent a SHOW_COLOR action, the color value would be set in an extra key-value pair.

Intent structure - Flags

Flags Field

- Flags of various sorts.
- Many instruct the Android system how to launch an activity (for example, which task the activity should belong to) and how to treat it after it's launched (for example, whether it belongs in the list of recent activities).

Intent Types

Types of Intents :

- Explicit intents
- Implicit intents

Intent Types - Explicit Intents

- Explicit Intents explicitly names the component which should be called by the Android system, by using the Java class as identifier.

```
Intent i = new Intent(this, ActivityTwo.class);
i.putExtra("Value1", "This value one for ActivityTwo ");
i.putExtra("Value2", "This value two ActivityTwo");
```

- Explicit Intents are typically used within on application as the classes in an application are controlled by the application developer.

Intent Types - Implicit Intents

- Implicit Intents do not specify the Java class which should be called. They specify the action which should be performed and optionally an URI which should be used for this action.

```
Intent intent = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.google.com"));
```

Intent Data Transfer

- An implicit Intent contains the Action and optional the URI. The receiving component can get this information via the getAction() and getData() methods.
- Explicit and implicit Intents can also contain additional data. This data call be filled by the component which creates the Intent. It can and can get extracted by the component which receives the Intent.

Intent Data Transfer

```
Intent sharingIntent = new Intent(Intent.ACTION_SEND);
sharingIntent.setType("text/plain");
sharingIntent.putExtra(android.content.Intent.EXTRA_TEXT, "News
    for you!");
// createChooser is a convenience method to create
// an Chooser Intent with a Title
startActivity(Intent.createChooser(sharingIntent,"Share this using"));
```

Intent Data Transfer

- The component which receives the Intent can use the getIntent().getExtras() method call to get the extra data.

```
Bundle extras = getIntent().getExtras();
if (extras == null) { return; }
 // Get data via the key
String value1 = extras.getString(Intent.EXTRA_TEXT);
if (value1 != null) {
// Do something with the data
 }
```

Sending Intent

- Sending Intent to start Activity：use method startActivity(Intent), startActivityForResult(Intent) of Context object
- Sending Intent to start/stop Service： use method startService (Intent)/stopService(Intent) of Context object
- Sending Intent to Broadcast receiver :use method sendBroadcast(Intent) of Context object

Receiving Intent

- Activity : use method getIntent() to get information of received Intent.
- Service: information of received Intent will be passed to method onStartCommand(Intent,int,int) of Service object
- BroadcastReceiver : information of received Intent will be passed to method onReceive(Context,Intent) of BroadcastReceiver object

Intent Filters

- What are Intent Filters?
  - Filters informs the system which implicit intents a component can handle
  - If a component does not have any intent filters, it can receive only explicit intents.
  - A component with filters can receive both explicit and implicit intents.
  - A component has separate filters for each job it can do

Intent Filters

- IntentFilters are typically defined via the AndroidManifest.xml file. For BroadcastReceiver it is also possible to define them in coding.
- An IntentFilters is defined by its category, action and data filters. It can also contain additional metadata.

Intent Filters

- An implicit Intent object are tested against an intent filters (of target components) in three areas
  - Action
  - Category
  - Data (both URI and data type)
- To be delivered to the target component that owns the filter, it must pass all three tests
- If an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate.
- If no target can be found, an exception is raised

Pending Intent

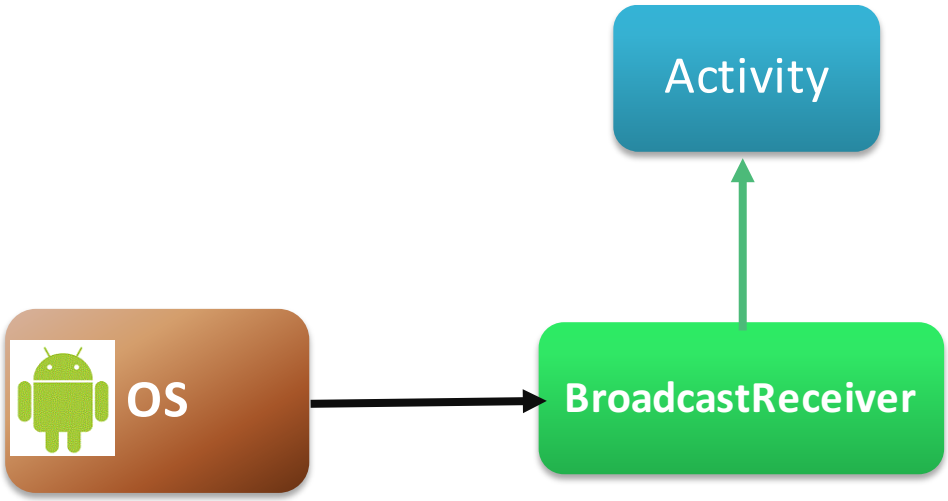
- A PendingIntent is a token that you give to another application (e.g. Notification Manager, Alarm Manager or other 3rd party applications), which allows this other application to use the permissions of your application to execute a predefined piece of code.
- To perform a broadcast via a pending intent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

    }

}
```

Pending Intent

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

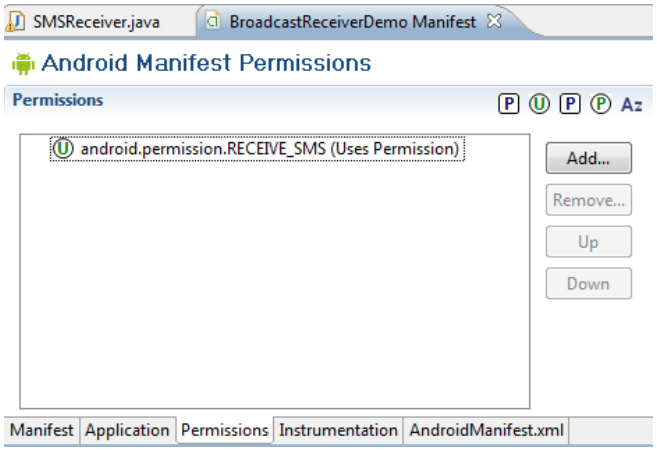
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.basistraining.broadcastreceiver" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
  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

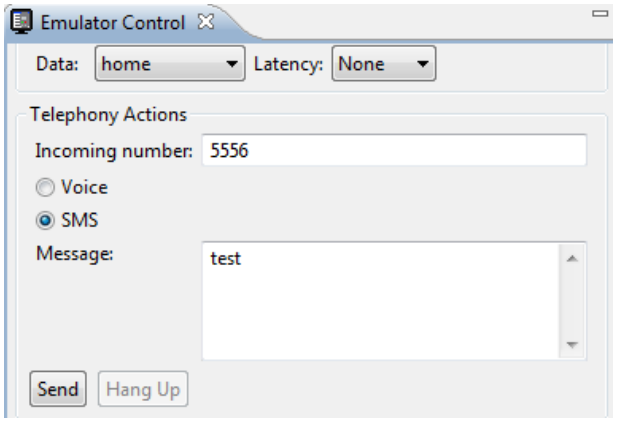


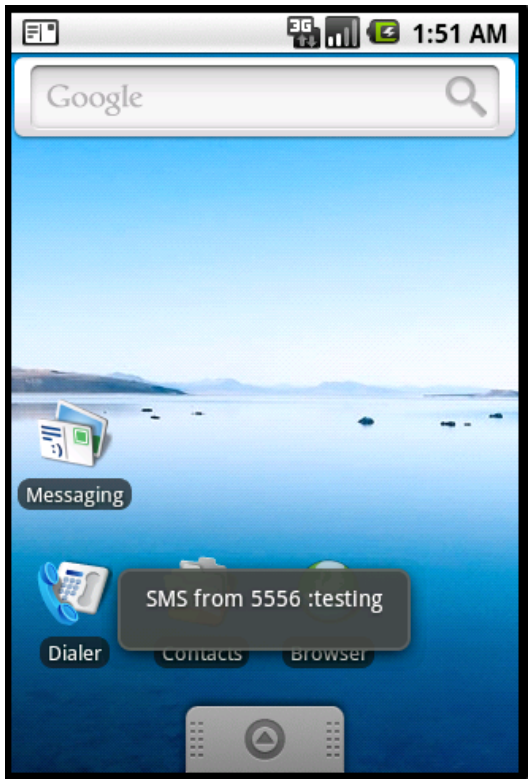

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.basistraining.broadcastreceiver" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
  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





Broadcast Receivers

```
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();
        }
```

Broadcast Receivers

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);
```

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

Exit Course

THANK YOU

EXIT