Queens University Belfast

Android Studio

CSC3054 / CSC7054

Week 4 Book 1 - Intents

Queens University Belfast

Introduction

An Android `Intent` is an abstract description of an operation to be performed. It can be used with `startActivity` to launch an Activity, `broadcastIntent` to send it to any interested `BroadcastReceiver` components, and `startService(Intent)` or `bindService(Intent, ServiceConnection, int)` to communicate with a background Service.

Starting an Activity

The simplest way one activity can start another is with the `Activity` method

```
public void startActivity(Intent intent)
```

You might guess that `startActivity(…)` is a class method that you call on the `Activity` subclass that you want to start. But it is not. When an activity calls `startActivity(…)`, this call is sent to the OS. In particular, it is sent to a part in the OS called the `ActivityManager`. The `ActivityManager` then creates the `Activity` instance and calls its `onCreate(…)` method.

How does the `ActivityManager` knows which `Activity` to start? That information is in the `Intent` parameter

Communicating with intents

An `intent` is an object that a component can use to communicate with the OS. The only components you have seen so far are activities, but there are also services, broadcast receivers, and content providers.

`Intents` are multi-purpose communication tools, and the `Intent` class provides different constructors depending on what you are using the intent to do.

For example, if you want to use the intent to tells the `ActivityManager` which activity to start you would use this constructor:

```
public Intent (Context packageContext, Class<?> cls)
```

The `Class` object specifies the activity that the `ActivityManager` should start. The `Context` object tells the `ActivityManager` which package the `Class` object can be found in.

Before starting the activity, the `ActivityManager` check's the package's `manifest` for a declaration with the same name as the specified `Class`. If it finds a declaration, it starts the activity, and all is well. If it does not, you get a nasty `ActivityNotFoundException`. This is why all of your activities must be declared in the `manifest`.

Explicit and implicit intents

Explicit Intents

When you create an `Intent` with a `Context` and a `Class` object, you are creating an `explicit` intent. You use explicit intents to start activities within your application. They are typically used for application-internal messages (Figure 1).

It may seem strange that two activities within your application must communicate via the `ActivityManager`, which is outside of your application. However, this pattern makes it easy for an activity in one application to work with an activity in another application.

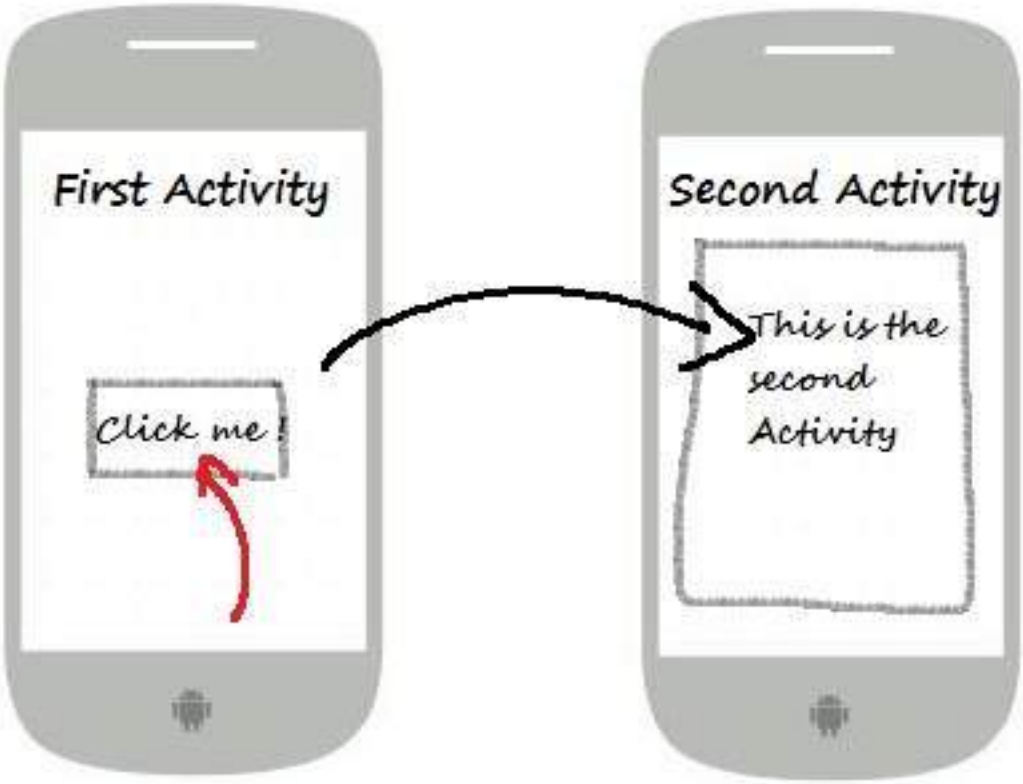

FIGURE 1 EXPLICIT INTENTS

The following shows how to create an explicit intent and send it to the Android system. If the class specified in the intent represents an activity, the Android system starts it.

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

Implicit intents

When an activity in your application wants to start an activity in another application, you create an implicit intent (figure 2). With an implicit intent you describe the job that you need done, and the OS will start an activity in an appropriate application for you.

For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter.

```
Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.bbc.co.uk"));
startActivity(i);
```

The OS starts the activity that has advertised itself as capable of doing that job. If the OS finds more than one capable activity, then the user is offered a choice.

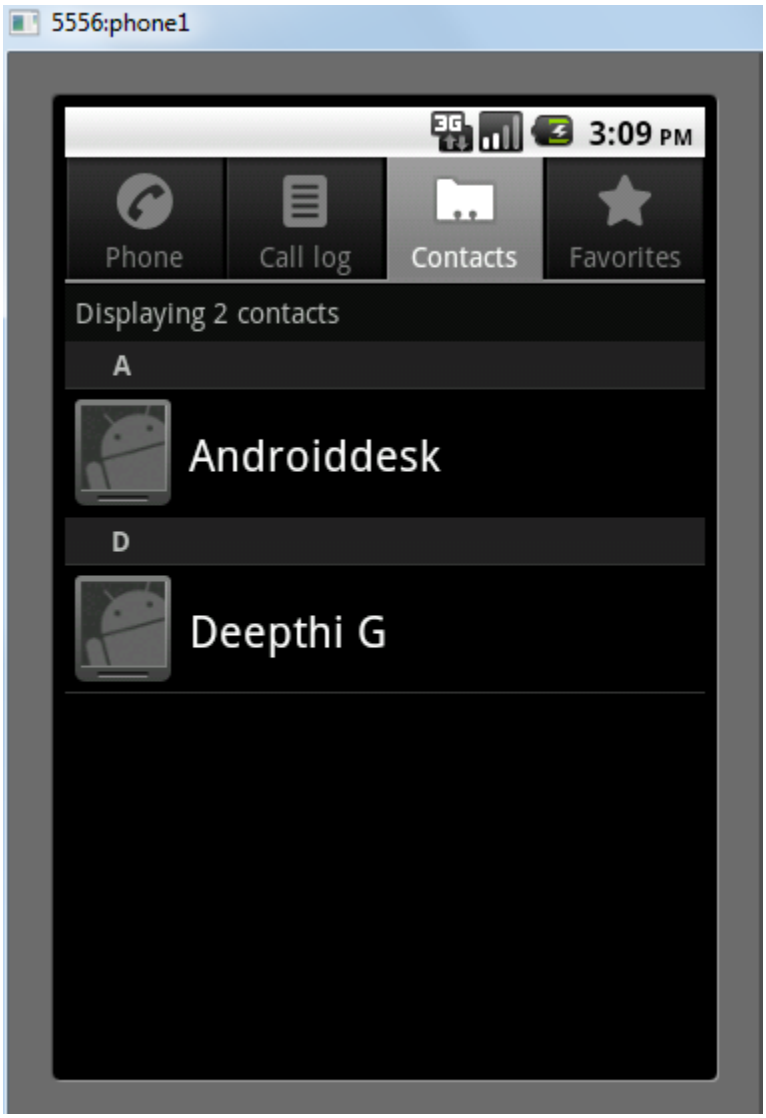

FIGURE 2 IMPLICIT INTENTS

Parts of an implicit intent

Here are the critical parts of an intent you can use to define the job you want done.

Part	Description
The `action` that you are trying to perform	These are typically `constants` from the `Intent` class. If you want to view a `URL`. You can use `Intent.ACTION_VIEW`. To send something, you use `Intent.ACTION_SEND`.
The location of any `data`	This can be something outside the device, like the `URL` of a web page, but it can also be a `URI` to a file or a `content URI` pointing to a record in a `ContentProvider`
The `type` of data that the action is for	This is a MIME type, like `text/html` or `audio/mp3`. If an `intent` includes a location for data, then the type can usually be inferred from that data.
Optional `categories`	If the action is used to describe what to do, the category usually describes `where, when` or `how` you are trying to use an activity. Android uses the category `android.intent.category.LAUNCHER` to indicate than an activity should be displayed in the top level app launcher. The android.intent.category.INFO category, on the other hand, indicates an activity that shows information about a package to the user but should not show up in the launcher.

A simple implicit intent for viewing a website should include an action of `Intent.ACTION_VIEW` and a data `Uri` that is the `URL` of a website.

Based on this information, the OS will launch the appropriate activity of an appropriate application. (If it finds more than one candidate, the user gets a choice).

An activity would advertise itself as an appropriate activity for `ACTION_VIEW` via an intent filter in the manifest. If you wanted to write a browser app, for instance, you would include the following intent filter in the declaration of the activity that should respond to `ACTION_VIEW`.

```
<activity>
android:name= ".BrowserActivity"
android:label= "@string/app-name">
<intent-filter>
      <action android: name =  "android.intent.action.VIEW"/>
      <category android: name "android.intent.category.DEFAULT"/>
      <data android: scheme = "http" android:host = "www.bbc.co.uk" />
</intent-filter>
</activity>
```

The `DEFAULT` category must be set explicitly in intent filters. The action element in the intent filter tells the OS that the activity is capable of performing the job, and the `DEFAULT` category tells the OS that it wants to be considered for the job. This `DEFAULT` category is implicitly added to almost every implicit intent. (The sole exception is the `LAUNCHER` category). Implicit intents can also include extra just like explicit intents. Any extras on an implicit intent, however, are not used by the OS to find an appropriate activity.

Note that the action and data parts of an intent class can also be used in conjunction with an explicit intent. That would be the equivalent of telling people a specific activity to do something particular.

Using intent extras

Optionally an intent can also contain additional data based on an instance of the `Bundle` class which can be retrieved from the intent via the `getExtras()` method. Extras are arbitrary data that the calling activity can include with an `intent`. You can also add data directly to the `Bundle` via the overloaded `putExtra()` methods of the `Intent` objects. The OS will then forward the intent to the recipient activity, which can then access the extra and retrieve the data. An extra is structured as a key value pair. To add an extra to an `intent`, you use `Intent.putExtra(…)`. In particular, you will be calling:

```
public Intent putExtra(String name, boolean value)
```

`Intent.putExtra(…)` comes in many flavors, but it always has two arguments. The first argument is always a `String` key, and the second argument is the value, whose type will vary. You can put multiple extras on an `Intent` if you need to. To retrieve the value from the extra, you will use

```
public Boolean getBooleanExtra(String name, Boolean defaultValue)
```

The first argument is the name of the extra. The second argument of `getBooleanExtra(…)` is a default answer if the key is not found. The receiving component can also access this information via the `getAction()` and `getData()` methods on the `Intent` object. This `Intent` object can be retrieved via the `getIntent()` method. The component which receives the intent can use the `getIntent().getExtras()` method call to get the extra data. That is demonstrated in the following:

```
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
}
```

Getting a result back from a child activity

When you want to hear back from the child activity, you call the following `Activity` method:

```
public void startActivityForResult(Intent intent, int requestCode)
```

The first parameter is the same intent as before. The second parameter is the `request code`. The `request code` is a user-defined integer that is sent to the child activity and then received back by the parent. It is used when an activity starts more than one type of child activity and needs to tell who is reporting back.

Setting a result

There are two methods you can call in the child activity to send data back to the parent:

```
Public final void setResult(int resultCode)

Public final void setResult(int resultCode, Intent data)
```

Typically, the result code is one of the two pre-defined contants: `Activity.RESULT_OK` or `Activity.RESULT_CANCELLED`. (You can use another constant, RESULT_FIRST_USER, as an offset when defining your own result codes.)

Setting result codes is useful when the parent needs to take a different action depending on how the child activity finished.

For example, if a child activity had an `OK` button and a `Cancel` button, the child activity would set a different result code depending on which button was pressed. The parent activity would take different action depending on the result code.

Calling `setResult(…)` is not required of the child activity. If you do not need to distinguish between results or receive arbitrary data on an intent, then you can let the OS send a default result code. A result code is always returned to the parent if the child activity was started with `startActivityForResult(…)`. If `setResult(…)` is not called, then, when the user presses the Back button, the parent will receive `Activity.RESULT_CANCELLED`.