# LAB 7 – Android Intents

At the end of this lab, students should be able to:

- Demonstrate the use of an Explicit Intent to launch an activity, including the transfer of data between sending and receiving activities.

## An overview of Intents

Intents (android.content.Intent) are the messaging system by which one activity is able to launch another activity. An activity can, for example, issue an intent to request the launch of another activity contained within the same application. Intents also, however, go beyond this concept by allowing an activity to request the services of any other appropriately registered activity on the device for which permissions are configured.
Consider, for example, an activity contained within an application that requires a web page to be loaded and displayed to the user. Rather than the application having to contain a second activity to perform this task, the code can simply send an intent to the
Android runtime requesting the services of any activity that has registered the ability to display a web page. The runtime system will match the request to available activities on the device and either launch the activity that matches or, in the event of multiple matches, allow the user to decide which activity to use.
Intents also allow for the transfer of data from the sending activity to the receiving activity. In the previously outlined scenario, for example, the sending activity would need to send the URL of the web page to be displayed to the second activity. Similarly, the receiving activity may also be configured to return data to the sending activity when the required tasks are completed. In addition to launching activities, intents are also used to launch and communicate with services and broadcast receivers.
Intents are categorized as either *explicit* or *implicit*.

### Explicit Interns

An explicit intent requests the launch of a specific activity by referencing the component name (which is actually the class name) of the target activity. This approach is most common when launching an activity residing in the same application as the sending activity (since the class name is known to the application developer).

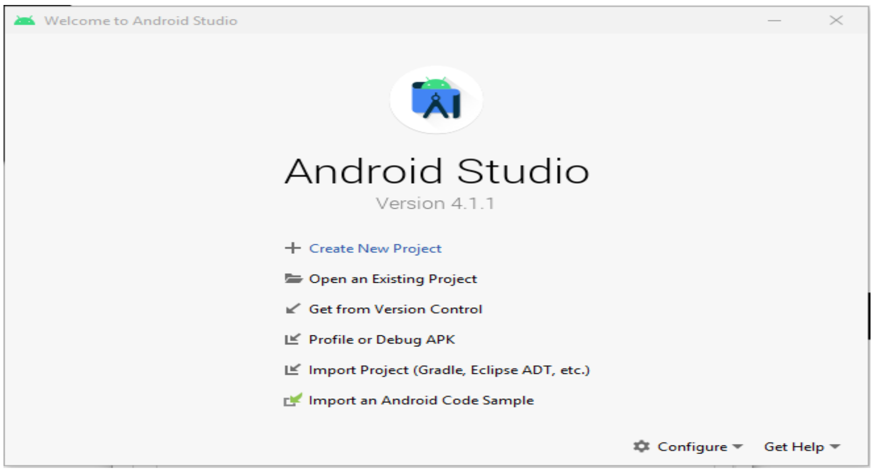
### Implicit Interns

Unlike explicit intents, which reference the class name of the activity to be launched, implicit intents identify the activity to be launched by specifying the action to be performed and the type of data to be handled by the receiving activity. For example, an action type of ACTION_VIEW accompanied by the URL of a web page in the form of a URI object will instruct the Android system to search for, and subsequently launch, a web browser capable activity.

# Android Explicit Intents Tutorial in Android Studio- An Example

## a) Creating the Explicit Intent Example Project

i. Go to File -> New Project



ii. Choose **Empty Activity** template, then click Next

iii. Enter *ExplicitIntent* into the Name field and specify com.ebookfrenzy.viewmodeldemo as the package name.
Change the Minimum API level setting to API26: Android 8.0(Oreo) and the Language menu to Java.
Click Finish.

## b) Designing the User Interface Layout for Main activity

Locate the *activity_main.xml* file in the Project tool window and double click on it to load it into the layout editor.

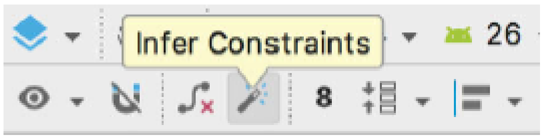
i. Select and delete the default “Hello World!” TextView.

ii. Drag a TextView widget from the palette and drop it so that it is centered within the layout and use the Attributes tool window to assign an ID of *textView1.*

iii. Drag a Button object from the palette and position it so that it is centered horizontally and located beneath the bottom edge of the TextView. Change the text property so that it reads “Send Text” and configure the *onClick* property to call a method named *sendText*.

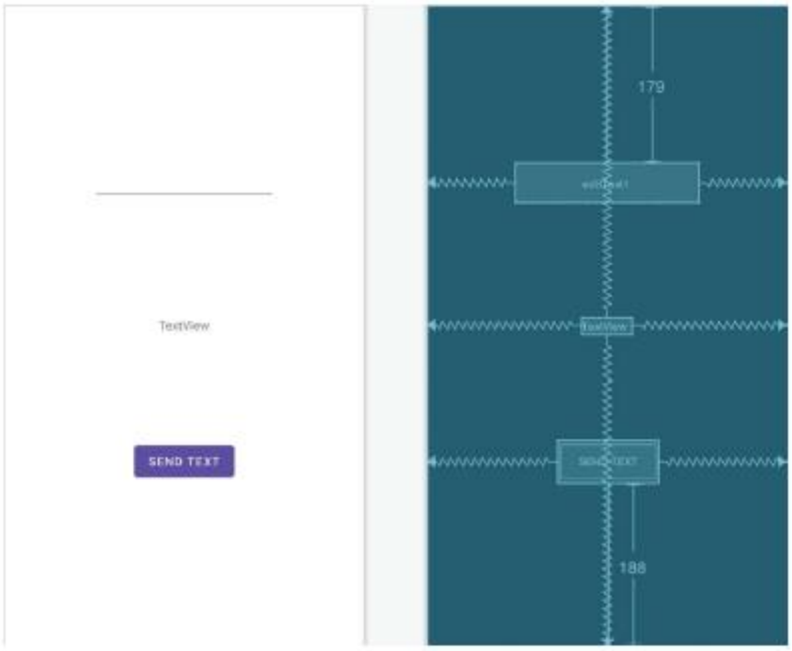
iv. Next, add a Plain Text object so that it is centered horizontally and positioned above the top edge of the TextView. Using the Attributes tool window, remove the “Name” string assigned to the text property and set the ID to *editText1*.

v. With the layout completed, click on the toolbar *Infer constraints* button to add appropriate constraints:



vi. Finally, click on the red warning button in the top right-hand corner of the Layout Editor window and use the resulting panel to extract the “Send Text” string to a resource named *send_text*.

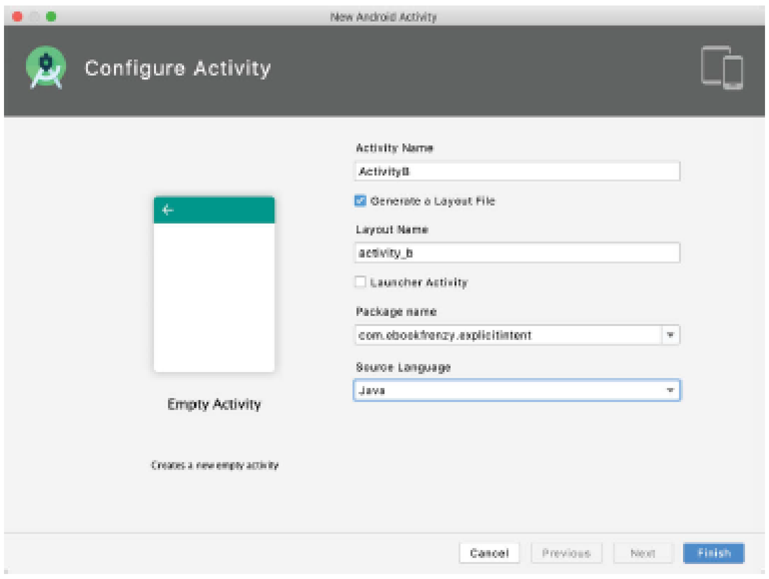
vii. Once the layout is complete, the user interface should resemble the figure below



## c) Creating the Second Activity Class

When the “Send Text” button is touched by the user, an intent will be issued requesting that a second activity be launched into which a response can be entered by the user.

i. The next step, therefore, is to create the second activity. Within the Project tool window, right-click on the *com.ebookfrenzy.explicitintent* package name located in app -> java and select *the New -> Activity -> Empty Activity* menu option to display the NewAndroid Activity dialog as shown below:


New Android Activity

Configure Activity

Activity Name: ActivityB

[✓] Generate a Layout File

Layout Name: activity_b

[ ] Launcher Activity

Package name: com.ebookfrenzy.explicitintent

Source Language: Java

Empty Activity

Creates a new empty activity

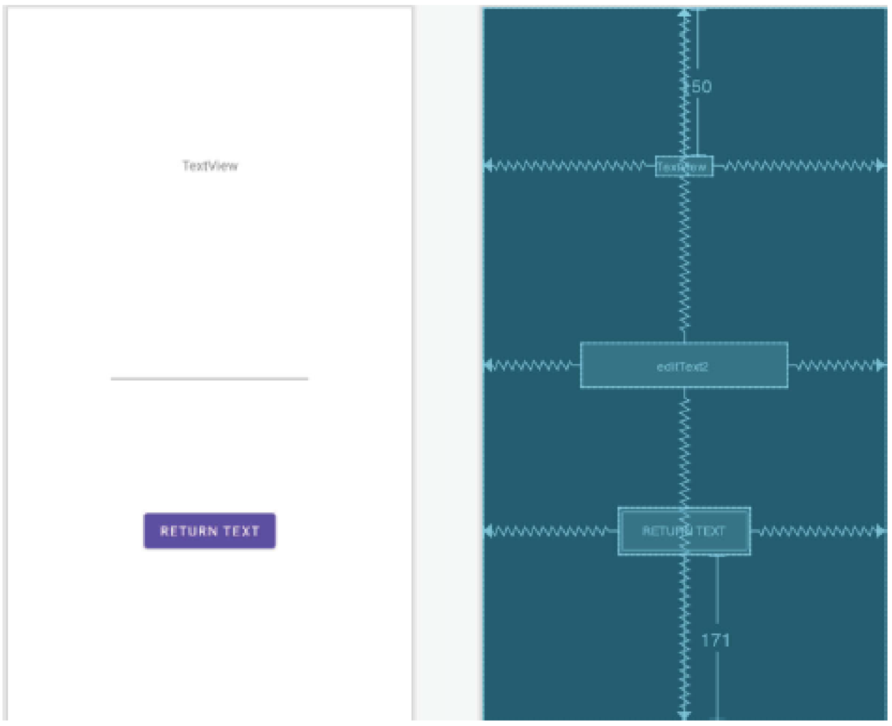
ii. Enter *ActivityB* into the Activity Name and Title fields and name the layout file *activity_b* and change the Language menu to Java. Since this activity will not be started when the application is launched (it will instead be launched via an intent by MainActivity when the button is pressed), it is important to make sure that the *Launcher Activity* option is disabled before clicking on the Finish button.

## d) Designing the User Interface Layout for Activity B

i. The elements that are required for the user interface of the second activity are a Plain Text EditText, TextView and Button view.

ii. With these requirements in mind, load the *activity_b.xml* layout into the Layout Editor tool, and add the views.

iii. During the design process, note that the *onClick* property on the button view has been configured to call a method named *returnText*, and the TextView and EditText views have been assigned IDs *textView2* and *editText2* respectively. Once completed, the layout should resemble that illustrated in figure below:



iv. Note that the text on the button (which reads “Return Text”) has been extracted to a string resource named *return_text*.

v. With the layout complete, click on the Infer constraints toolbar button to add the necessary constraints to the layout:

## e) Reviewing the Application Manifest File

In order for MainActivity to be able to launch ActivityB using an intent, it is necessary that an entry for ActivityB be present in the *AndroidManifest.xml* file.

i. Locate this file within the Project tool window (*app -> manifests*), double-click on it to load it into the editor and verify that Android Studio has automatically added an entry for the activity:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.ebookfrenzy.explicitintent">
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<activity android:name=".ActivityB"></activity>
<activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN"
/>
<category
android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>
</manifest>
```

With the second activity created and listed in the manifest file, it is now time to write some code in the MainActivity class to issue the intent.

## f) Creating the Intent

i. The objective for MainActivity is to create and start an intent when the user touches the “Send Text” button. As part of the intent creation process, the question string entered by the user into the EditText view will be added to the intent object as a key-value pair.

ii. When the user interface layout was created for MainActivity, the button object was configured to call a method named *sendText()* when “clicked” by the user. This method now needs to be added to the MainActivity class *MainActivity.java* source file as follows:

```
package com.ebookfrenzy.explicitintent;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_a);
}
public void sendText(View view) {
Intent i = new Intent(this, ActivityB.class);
final EditText editText1 = findViewById(R.id.editText1);
String myString = editText1.getText().toString();
i.putExtra("qString", myString);
startActivity(i);
}
}
```

The explanation for the code for the *sendText()* method as follows:
First, a new Intent instance is created, passing through the current activity and the class name of ActivityB as arguments. Next, the text entered into the EditText object is added to the intent object as a key-value pair and the intent started via a call to *startActivity(),* passing through the intent object as an argument.

iii. Compile and run the application and touch the “Send Text” button to launch ActivityB and the back button (located in the toolbar along the bottom of the display) to return to MainActivity.

## g) Extracting Intent Data

i. Now that ActivityB is being launched from MainActivity, the next step is to extract the String data value included in the intent and assign it to the TextView object in the ActivityB user interface. This involves adding some code to the onCreate() method of ActivityB in the *ActivityB.java* source file:

```
package com.ebookfrenzy.explicitintent;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.TextView;
import android.widget.EditText;
public class ActivityB extends AppCompatActivity {
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activityb);
Bundle extras = getIntent().getExtras();
if (extras == null) {
return;
}
String qString = extras.getString("qString");
final TextView textView =
findViewById(R.id.textView2);
textView.setText(qString);
}
}
```

ii. Compile and run the application either within an emulator or on a physical Android device. Enter a question into the text box in MainActivity before touching the “Send Text” button. The message should now appear on the TextView component in the ActivityB user interface.

## h) Launching ActivityB as a Sub-Activity

In order for ActivityB to be able to return data to MainActivity, ActivityB must be started as a *sub-activity* of MainActivity. This means that the call to *startActivity()* in the MainActivity *returnText()* method needs to be replaced with a call to startActivityForResult(). Unlike the *startActivity()* method, which takes only the intent object as an argument, *startActivityForResult()* requires that a request code also be passed through. _e request code can be any number value and is used to identify which sub-activity is associated with which set of return data.

i. For the purposes of this example, a request code of 5 will be used, giving us a modified MainActivity class that reads as follows:

```
public class MainActivity extends AppCompatActivity {
private static final int request_code = 5;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
public void sendText(View view) {
Intent i = new Intent(this, ActivityB.class);
final EditText editText1 =
findViewById(R.id.editText1);
String myString = editText1.getText().toString();
i.putExtra("qString", myString);
~~startActivity(i);~~
startActivityForResult(i, request_code);
}
}
```

ii. When the sub-activity exits, the *onActivityResult()* method of the parent activity is called and passed as arguments the request code associated with the intent, a result code indicating the success or otherwise of the sub-activity and an intent object containing any data returned by the sub-activity. Remaining within the MainActivity class source file, implement this method as follows:

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if ((requestCode == request_code) &&
(resultCode == RESULT_OK)) {
TextView textView1 = findViewById(R.id.textView1);
String returnString =
data.getExtras().getString("returnData");
textView1.setText(returnString);
}
}
```

The code in the above method begins by checking that the request code matches the one used when the intent was issued and ensuring that the activity was successful. The return data is then extracted from the intent and displayed on the TextView object.

## i) Returning Data from a Sub-Activity

ActivityB is now launched as a sub-activity of MainActivity, which has, in turn, been modified to handle data returned from ActivityB. All that remains is to modify *ActivityB.java* to implement the *finish()* method and to add code for the *returnText()* method, which is called when the “Return Text” button is touched. The *finish()* method is triggered when an activity exits (for example when the user selects the back button on the device):

```
public void returnText(View view) {
                finish();
}
@Override
public void finish() {
Intent data = new Intent();
EditText editText2 = findViewById(R.id.editText2);
String returnString = editText2.getText().toString();
data.putExtra("returnData", returnString);
setResult(RESULT_OK, data);
super.finish();
}
```

All that the *finish()* method needs to do is create a new intent, add the return data as a key-value pair and then call the *setResult()* method, passing through a result code and the intent object. The *returnText()* method simply calls the *finish()* method.

## j) Testing the application

Compile and run the application, enter a question into the text field on MainActivity and touch the “Send Text” button. When ActivityB appears, enter the text to the EditText view and use either the back button or the “Return Text” button to return to MainActivity where the response should appear in the text view object.