COMP3040 Lab Session L02 – Intents

Explicit Intents

The exercise will demonstrate the use of an explicit intent to launch an activity, including the transfer of data between sending and receiving activities.

Launch Android Studio and create a new project, entering *ExplicitIntent* into the Application name field and *example.com* as the Company Domain setting before clicking on the Next button.

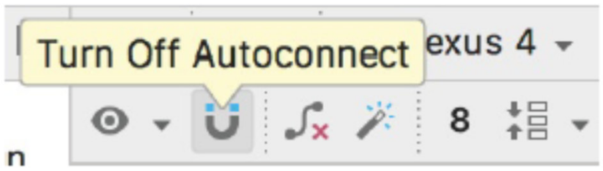
On the form factors screen, enable the Phone and Tablet option and set the minimum SDK setting to API 19: Android 4.4 (KitKat).

Continue to proceed through the screens, requesting the creation of an Empty Activity named *ActivityA* with a corresponding layout named *activity_a*.

Click Finish to create the new project.

The user interface for ActivityA will consist of a ConstraintLayout view containing EditText (Plain Text), TextView and Button views named editText1, textView1 and button1 respectively. Using the Project tool window, locate the activity_a.xml resource file for ActivityA (located under app -> res -> layout) and double-click on it to load it into the Android Studio Layout Editor tool. Select and delete the default "Hello World!" TextView.

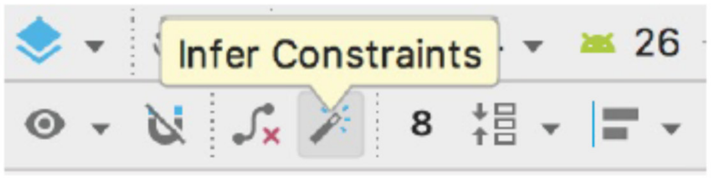
For this exercise, Inference mode will be used to add constraints after the layout has been designed. Begin, therefore, by turning off the Autoconnect feature of the Layout Editor using the toolbar button:



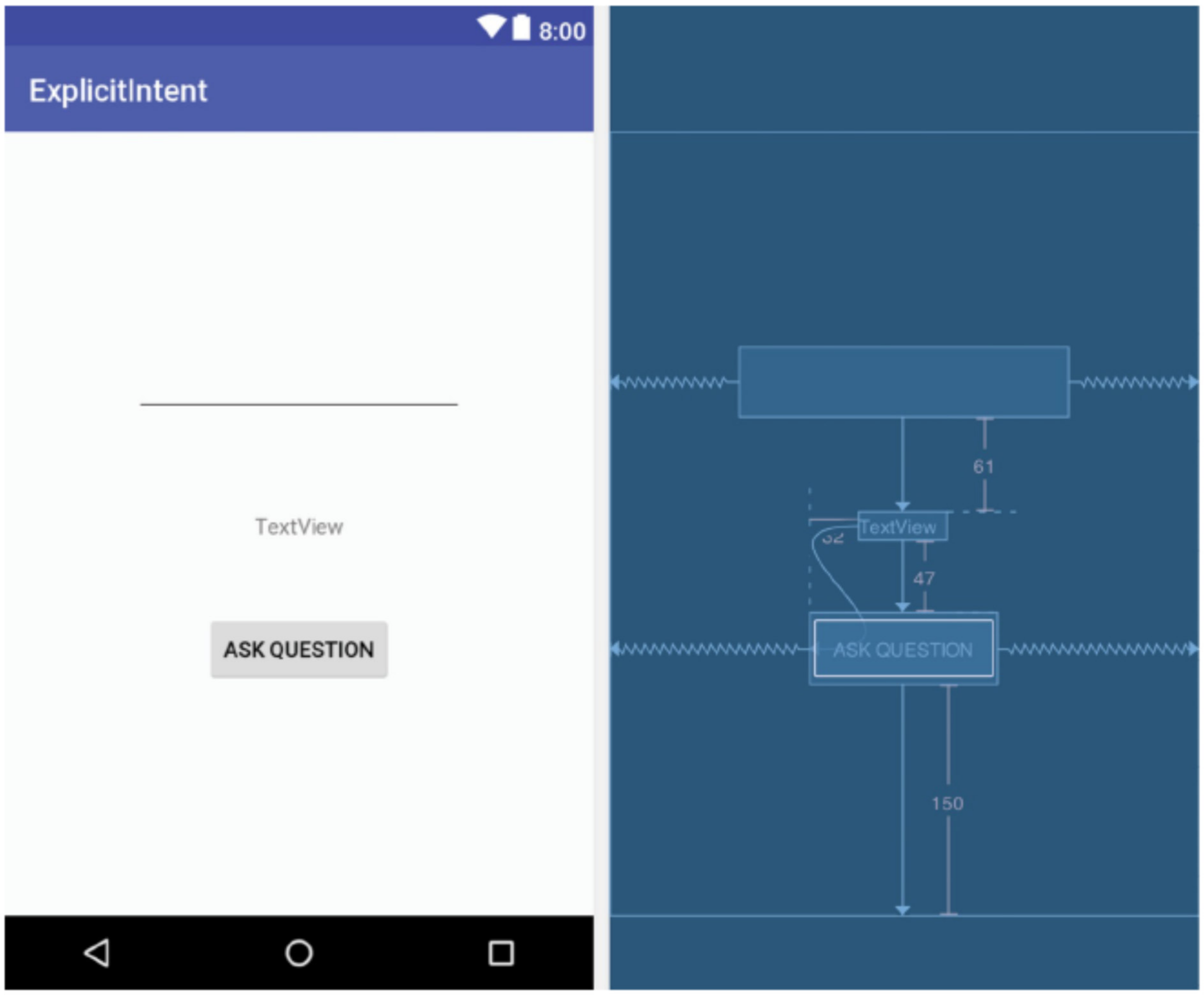
Drag a TextView widget from the palette and drop it so that it is centered within the layout and use the Attributes tool window to assign an ID of textView1.

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 "Ask Question" and configure the onClick property to call a method named onClick().

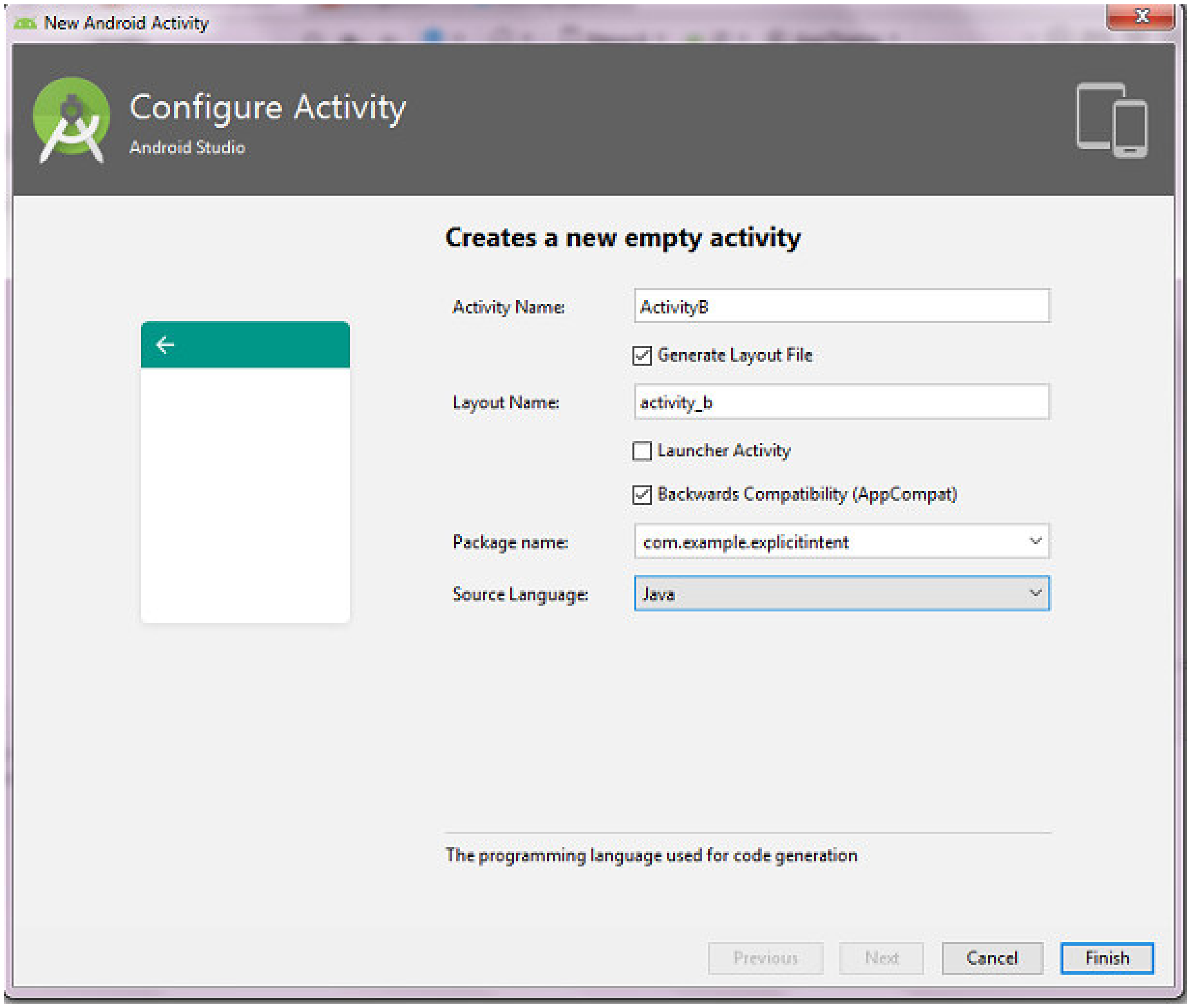
Next, add an Plain Text object so that it is centered horizontally and positioned above the top edge of the TextView. Using the Attributes tool window, remove the "Name" string assigned to the text property and set the ID to editText1. With the layout completed, click on the toolbar Infer constraints button to add appropriate constraints:



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 “Ask Question” string to a resource named ask_question:



When the “Ask Question” button is touched by the user, an intent will be issued requesting that a second activity be launched into which an answer can be entered by the user. The next step, therefore, is to create the second activity. Within the Project tool window, right-click on the com.example.explicitintent package name located in app -> java and select the New -> Activity -> Empty Activity menu option to display the New Android Activity dialog:


New Android Activity

Configure Activity

Android Studio

Creates a new empty activity

Activity Name: ActivityB

[✓] Generate Layout File

Layout Name: activity_b

[ ] Launcher Activity

[✓] Backwards Compatibility (AppCompat)

Package name: com.example.explicitintent

Source Language: Java

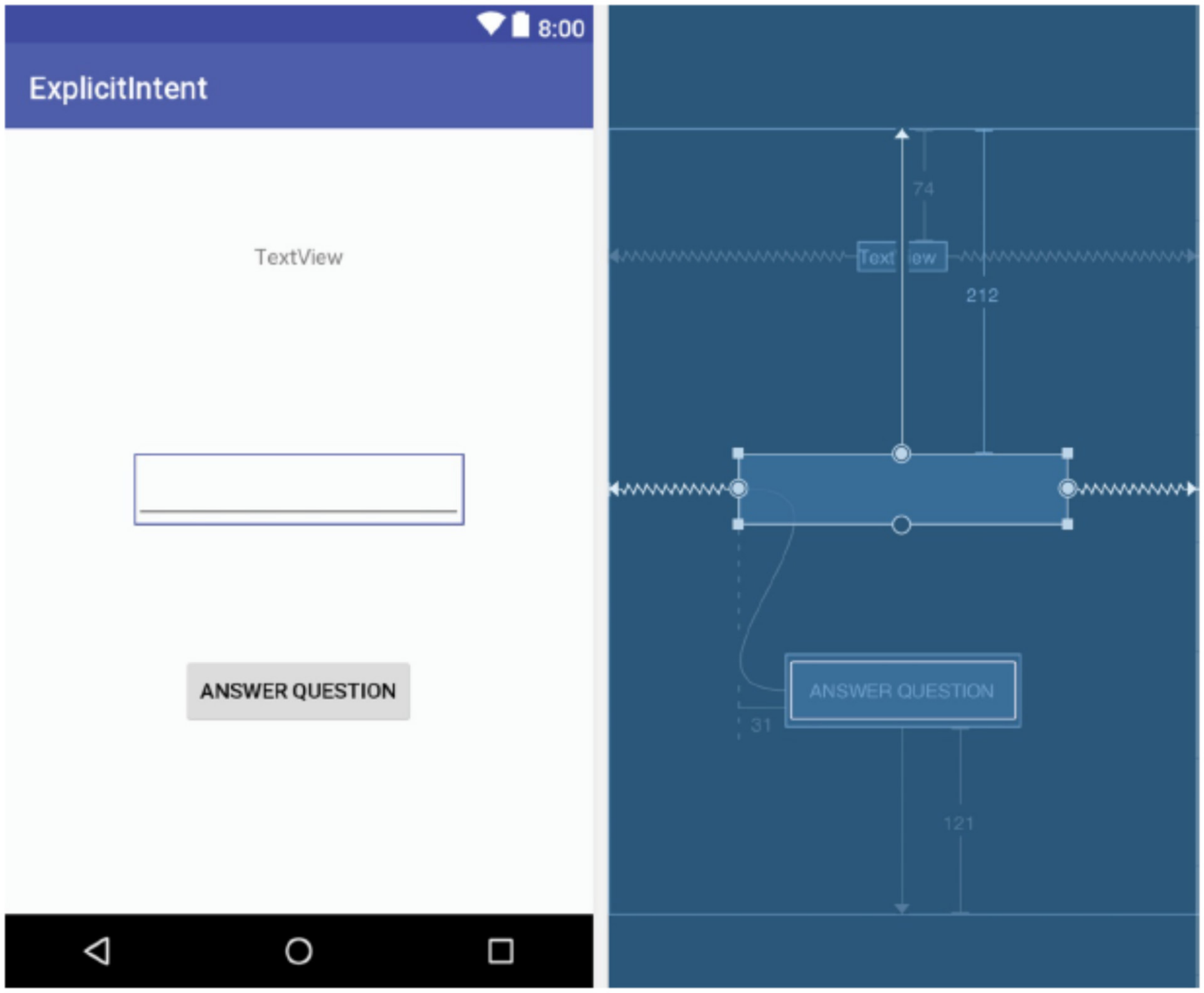
The programming language used for code generation

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

The elements that are required for the user interface of the second activity are a Plain Text EditText, TextView and Button view. With these requirements in mind, load the activity_b.xml layout into the Layout Editor tool, turn off Autoconnect mode in the Layout Editor toolbar and add the views.

During the design process, note that the onClick property on the button view has been configured to call a method named onClick(), and the TextView and EditText views have been assigned IDs textView1 and editText1 respectively. Note that the text on the button (which reads "Answer Question") has been extracted to a string resource named answer_question.

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



In order for ActivityA to be able to launch ActivityB using an intent, it is necessary that an entry for ActivityB be present in the AndroidManifest.xml file. 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.example.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=".ActivityA">
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ActivityB"></activity>
    </application>

</manifest>
```

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

The objective for ActivityA is to create and start an intent when the user touches the "Ask Question" 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. When the user interface layout was created for ActivityA, the button object was configured to call a method named onClick() when "clicked" by the user. This method now needs to be added to the ActivityA class ActivityA.java source file as follows:

```
package com.example.explicitintent;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class ActivityA extends AppCompatActivity {

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

    public void onClick(View view) {

        Intent i = new Intent(this, ActivityB.class);

        final EditText editText1 = (EditText)
                findViewById(R.id.editText1);
        String myString = editText1.getText().toString();
        i.putExtra("qString", myString);
        startActivity(i);
    }
}
```

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 keyvalue pair and the intent started via a call to startActivity(), passing through the intent object as an argument.

Compile and run the application and touch the “Ask Question” button to launch ActivityB and the back button (located in the toolbar along the bottom of the display) to return to ActivityA.

Now that ActivityB is being launched from ActivityA, 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.example.explicitintent;

import android.support.v7.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 {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_b);

        Bundle extras = getIntent().getExtras();
        if (extras == null) {
            return;
        }

        String qString = extras.getString("qString");

        final TextView textView = (TextView)
                findViewById(R.id.textView1);
        textView.setText(qString);
    }
}
```

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

In order for ActivityB to be able to return data to ActivityA, ActivityB must be started as a sub-activity of ActivityA. This means that the call to startActivity() in the ActivityA onClick() 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. The request code can be any number value and is used to identify which sub-activity is associated with which set of return data. For the purposes of this example, a request code of 5 will be used, giving us a modified ActivityA class that reads as follows:

```
public class ActivityA extends AppCompatActivity {

    private static final int request_code = 5;

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

    public void onClick(View view) {

        Intent i = new Intent(this, ActivityB.class);

        final EditText editText1 = (EditText)
                findViewById(R.id.editText1);
        String myString = editText1.getText().toString();
        i.putExtra("qString", myString);
        startActivityForResult(i, request_code);
    }
}
```

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 ActivityA class source file, implement this method as follows:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if ((requestCode == request_code) &&
            (resultCode == RESULT_OK)) {

        TextView textView1 =
                (TextView) 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.

ActivityB is now launched as a sub-activity of ActivityA, 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 onClick() method, which is called when the “Answer Question” 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 onClick(View view) {
    finish();
}

@Override
public void finish() {
    Intent data = new Intent();

    EditText editText1 = (EditText) findViewById(R.id.editText1);

    String returnString = editText1.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 onClick() method simply calls the finish() method.

Compile and run the application, enter a question into the text field on ActivityA and touch the “Ask Question” button. When ActivityB appears, enter the answer to the question and use either the back button or the “Submit Answer” button to return to ActivityA where the answer should appear in the text view object.

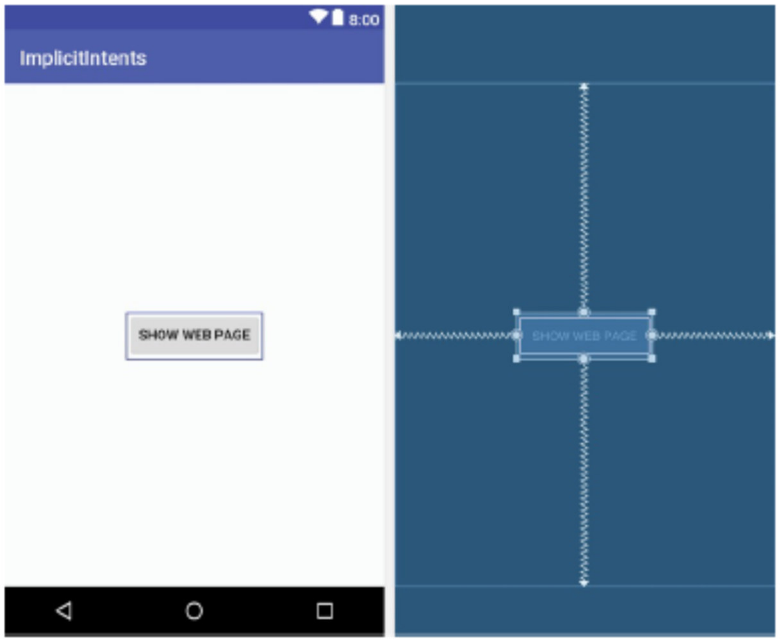
Implicit Intents

Launch Android Studio and create a new project, entering ImplicitIntent into the Application name field and example.com as the Company Domain setting before clicking on the Next button.

On the form factors screen, enable the Phone and Tablet option and set the minimum SDK to API 19: Android 4.4 (KitKat). Continue to proceed through the screens, requesting the creation of an Empty Activity named ImplicitIntentActivity with a corresponding layout resource file named activity_implicit_intent. Click Finish to create the new project.

The user interface for the ImplicitIntentActivity class is very simple, consisting solely of a ConstraintLayout and a Button object. Within the Project tool window, locate the app -> res -> layout -> activity_implicit_intent.xml file and double-click on it to load it into the Layout Editor tool.

Delete the default TextView and, with Autoconnect mode enabled, position a Button widget so that it is centered within the layout. Note that the text on the button ("Show Web Page") has been extracted to a string resource named show_web_page.



With the Button selected use the Attributes tool window to configure the onClick property to call a method named showWebPage().

As outlined above, the implicit intent will be created and issued from within a method named showWebPage() which, in turn, needs to be implemented in the ImplicitIntentActivity class, the code for which resides in the ImplicitIntentActivity.java source file. Locate this file in the Project tool window and double-click on it to load it into an editing pane. Once loaded, modify the code to add the showWebPage() method together with a few requisite imports:

The tasks performed by this method are actually very simple. First, a new intent object is created. Instead of specifying the class name of the intent, however, the code simply indicates the nature of the intent (to display something to the user) using the ACTION_VIEW option. The intent object also includes a URI containing the URL to be displayed. This indicates to the Android intent resolution system that the activity is requesting that a web page be displayed. The intent is then issued via a call to the startActivity() method.

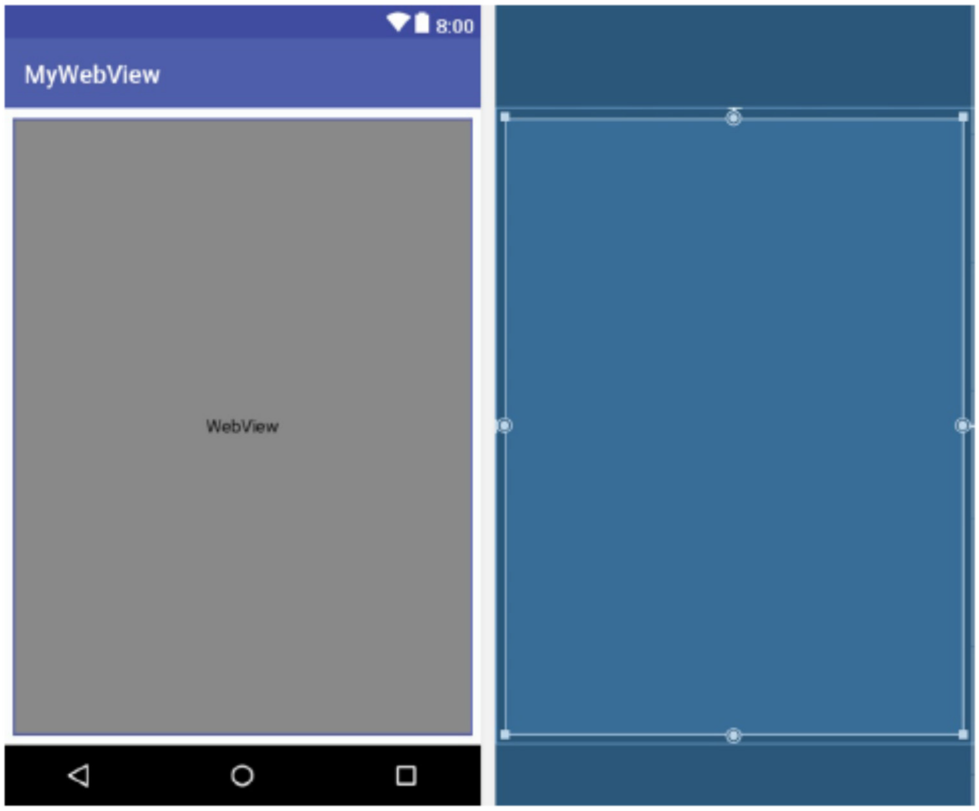
Compile and run the application on either an emulator or a physical Android device and, once running, touch the Show Web Page button. When touched, a web browser view should appear and load the web page designated by the URL. A successful implicit intent has now been executed.

The following exercise will demonstrate the effect of the presence of more than one activity installed on the device matching the requirements for an implicit intent. To achieve this, a second application will be created and installed on the device or emulator. Begin, therefore, by creating a new project within Android Studio with the application name set to MyWebView, using the same SDK configuration options used when creating the ImplicitIntent project earlier in this exercise. Select an Empty Activity and, when prompted, name the activity MyWebViewActivity and the layout and resource file activity_my_web_view.

The user interface for the sole activity contained within the new MyWebView project is going to consist of an instance of the Android WebView widget. Within the Project tool window, locate the activity_my_web_view.xml file, which contains the user interface description for the activity, and double-click on it to load it into the Layout Editor tool.

With the Layout Editor tool in Design mode, select the default TextView widget and remove it from the layout by using the keyboard delete key.

Drag and drop a WebView object from the Containers section of the palette onto the existing ConstraintLayout view:



Before continuing, change the ID of the WebView instance to webView1.

When the implicit intent object is created to display a web browser window, the URL of the web page to be displayed will be bundled into the intent object within a Uri object. The task of the onCreate() method within the MyWebViewActivity class is to extract this Uri from the intent object, convert it into a URL string and assign it to the WebView object. To implement this functionality, modify the MyWebViewActivity.java file so that it reads as follows:

```
package com.example.mywebview;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import java.net.URL;
import android.net.Uri;
import android.content.Intent;
import android.webkit.WebView;

public class MyWebViewActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my_web_view);

        handleIntent();
    }

    private void handleIntent() {
        Intent intent = getIntent();
        Uri data = intent.getData();
        URL url = null;
        try {
            url = new URL(data.getScheme(),
                    data.getHost(),
                    data.getPath());
        } catch (Exception e) {
            e.printStackTrace();
        }
        WebView webView = (WebView) findViewById(R.id.webView1);
        webView.loadUrl(url.toString());
    }
}
```

The new code added to the onCreate() method performs the following tasks:

- Obtains a reference to the intent which caused this activity to be launched
- Extracts the Uri data from the intent object
- Converts the Uri data to a URL object
- Obtains a reference to the WebView object in the user interface
- Loads the URL into the web view, converting the URL to a String in the process

The coding part of the MyWebView project is now complete. All that remains is to modify the manifest file.

There are a number of changes that must be made to the MyWebView manifest file before it can be tested. In the first instance, the activity will need to seek permission to access the internet (since it will be required to load a web page). This is achieved by adding the appropriate permission line to the manifest file:

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

Further, a review of the contents of the intent filter section of the AndroidManifest.xml file for the MyWebView project will reveal the following settings:

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

In the above XML, the android.intent.action.MAIN entry indicates that this activity is the point of entry for the application when it is launched without any data input. The android.intent.category.LAUNCHER directive, on the other hand, indicates that the activity should be listed within the application launcher screen of the device.

Since the activity is not required to be launched as the entry point to an application, cannot be run without data input (in this case a URL) and is not required to appear in the launcher, neither the MAIN nor LAUNCHER directives are required in the manifest file for this activity.

The intent filter for the MyWebViewActivity activity does, however, need to be modified to indicate that it is capable of handling ACTION_VIEW intent actions for http data schemes.

Android also requires that any activities capable of handling implicit intents that do not include MAIN and LAUNCHER entries, also include the so-called default category in the intent filter. The modified intent filter section should therefore read as follows:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:scheme="http" />
</intent-filter>
```

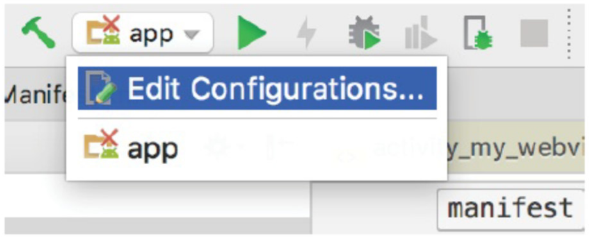
Bringing these requirements together results in the following complete AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mywebview" >

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

    <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=".MyWebViewActivity" >
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:scheme="http" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

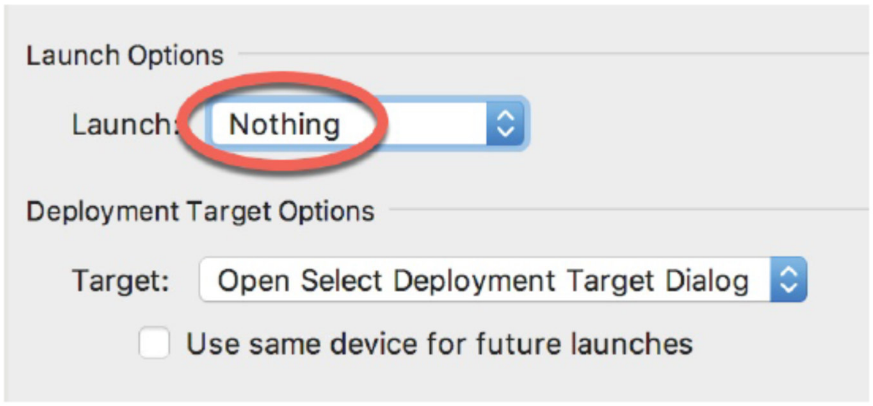
Load the AndroidManifest.xml file into the manifest editor by double-clicking on the file name in the Project tool window. Once loaded, modify the XML to match the above changes.

Having made the appropriate modifications to the manifest file, the new activity is ready to be installed on the device.

Before the MyWebViewActivity can be used as the recipient of an implicit intent, it must first be installed onto the device. This is achieved by running the application in the normal manner. Because the manifest file contains neither the android.intent.action.MAIN nor the android.intent.category.LAUNCHER Android Studio needs to be instructed to install, but not launch, the app. To configure this behavior, select the app -> Edit configurations… menu from the toolbar:



Within the Run/Debug Configurations dialog, change the Launch option located in the Launch Options section of the panel to Nothing and click on Apply followed by OK:


Launch Options

Launch: Nothing

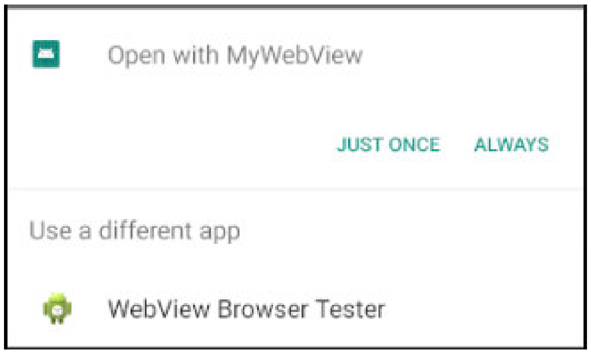
Deployment Target Options

Target: Open Select Deployment Target Dialog

Use same device for future launches

With this setting configured run the app as usual. Note that the app is installed on the device, but not launched.

In order to test MyWebView, simply re-launch the ImplicitIntent application created earlier in this exercise and touch the Show Web Page button. This time, however, the intent resolution process will find two activities with intent filters matching the implicit intent. As such, the system will display a dialog providing the user with the choice of activity to launch.



Selecting the MyWebView option followed by the Just once button should cause the intent to be handled by our new MyWebViewActivity, which will subsequently appear and display the designated web page.