# Lab – Intents

Objectives:

Familiarize yourself with Android Intents. You will create Intents and then use them to activate Activities.

The Intent class

In this lab, you will use both explicit and implicit Intents to activate Activities. In one case, you will start an Activity using the startActivity() method. In another case, you will start an Activity by using startActivityForResult(). See http://developer.android.com/training/basics/intents/ for more information.

The main Activity for this application is called ActivityLoaderActivity. It should display two Buttons, one labeled "Explicit Activation" and one labeled "Implicit Activation." It should also display one TextView, initially displaying the words, "No Text Entered".
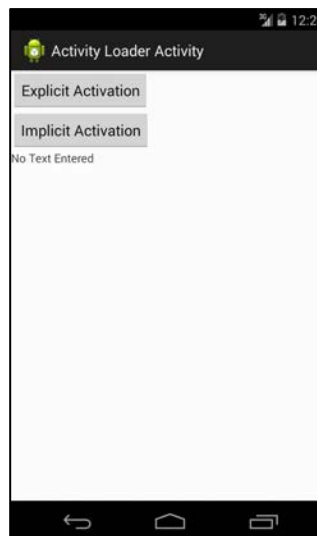
The application should behave as follows. When the user clicks the "Explicit Activation" Button, the ActivityLoaderActivity should launch a new Activity called ExplicitlyLoadedActivity. This activity should display a user interface containing an EditText and a Button. An EditText object allows a user to enter text.

When the user clicks the "Enter" Button, this Activity should finish, returning the current contents of the EditText object (seen above as 'Hello World!) back to the ActivityLoaderActivity. Once the ActivityLoaderActivity is again visible, any text that was returned from the ExplicitlyLoadedActivity should appear in the ActivityLoaderActivity's TextView object, as shown below.
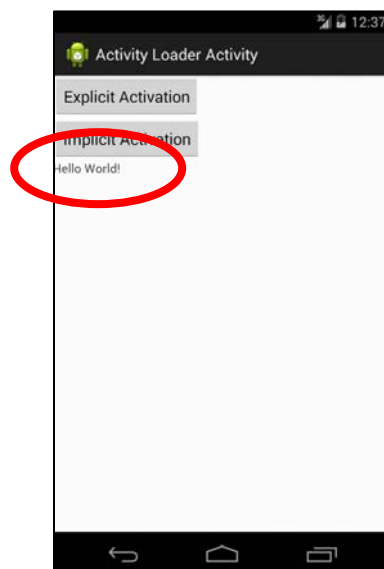
When the user clicks on the "Implicit Activation" Button, the ActivityLoaderActivity will create an implicit Intent, and then use it to implicitly activate a separate application to view the URL, "http://www.google.com". Because multiple applications may be able to handle this Intent, ActivityLoaderActivity will create and display an App Chooser (one example is shown below, but yours may have a slightly different layout), allowing the user to select the specific application to handle the Intent. For this assignment, the Chooser should display two choices: Android's built-in Browser and a separate application you've created called MyBrowser. To create an App Chooser, start by creating an initial Intent to view a webpage, (as part of this process you'll need to use the Uri class' parse() method). Then, create a second Intent, based on the first one, by calling the Intent class' createChooser() method. Finally, start a new Activity using this second Intent. See http://developer.android.com/training/basics/intents/sending.html for more information about creating chooser Intents.
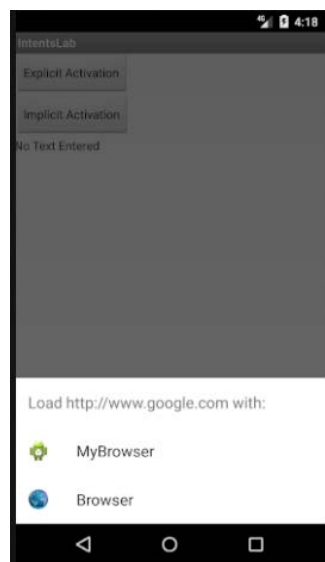
If the user selects the Android Browser from this chooser dialog, then that application will open and display the webpage at the given URL. If the user instead selects the MyBrowser application, then that application will open and simply display the text of the URL in a TextView.

See the screencast in the download package to see the app in action.

Implementation Notes:

Import the two projects Lab2_Intents and Lab2_MyBrowser. For the Lab2_Intents project, perform the following steps.

1) In ActivityLoaderActivity.java, implement launching the ExplicitlyLoadedActivity and returning the text result to the ActivityLoaderActivity.

   a) When the user clicks the "Explicit Activation" Button create an explicit Intent for launching the ExplicitlyLoadedActivity.

   b) In the ActivityLoaderActivity onActivityResult() will eventually be called. In that method you should use Intent.getStringExtra() to read it back out. See http://developer.android.com/training/basics/intents/result.html for more information.

2) In ExplicitlyLoadedActivity.java, finish the Activity and return any user-entered text when the user clicks the Enter Button.

   a) To return the text, you can use the Intent.putExtra() method to store the text in an Intent to be returned to the ActivityLoaderActivity. What key will you use when storing the text? You will pass this Intent to the setResult() method, add an appropriate result code, and then call finish().

3) In ActivityLoaderActivity, implement launching an unknown Activity to view a URL.

   a) When the "Implicit Activation" Button is clicked you should create an implicit Intent indicating that you want to a view a URL. The application should use an app chooser to display Activities that are capable of viewing URLs. See http://developer.android.com/training/basics/intents/sending.html for more information.

For the Lab2_MyBrowser project, do the following.

4) Implement and deploy the Lab2_MyBrowser application.

   a) In Lab2_MyBrowser's AndroidManifest.xml file, add the appropriate Intent Filter to MyBrowserActivity so that Android will know that this Activity can view web pages. To indicate that the Activity can handle this type of Intent, you will need to add an <intent-filter> to the <activity> in Lab2_MyBrowser's AndroidManifest.xml file. Be sure to add the correct <action>, <category>, and <data> elements to an <intent-filter> element.

As explained above, you can find the skeleton code in the Lab's download package. Below are the list of classes and methods you need to implement. You can find them in the code by looking for comments with the String TODO:

1.   In ActivityLoaderActivity.java implement the following methods.

        private void startExplicitActivation(): create an intent and start the ExplicitlyLoadedActivity.

private void startImplicitActivation(): create an Implicit Intent in **public Intent** getBaseIntent() and use the Intent.createChooser() method to select an application for viewing a web page. What action string should you use? Activate the user-selected Activity using this Implicit Intent.

protected void onActivityResult(**int** requestCode, **int** resultCode, Intent data): get the data that is returned from the ExplicitlyLoadedActivity Activity and display it in the mUserTextView. How do you find and access that TextView?

2.   In ExplicitlyLoadedActivity.java implement the following method.

private void enterClicked(): get user-entered text from the EditText, store it in an Intent, and return it to the ActivityLoaderActivity.

3.   In Lab2_MyBrowser's AndroidManifest.xml file, add the necessary intent filter tags. You'll need to define the right <action>, <category> and <data> elements. What values should be used for these? Remember that how you express certain values in Java can be different than how you express them in xml.

Testing

The test cases for this Lab are in the Lab2_Intents.. You can run the test cases either all at once, by right clicking on the tests directory and then selecting Run 'Tests in 'course.lab…', or one at a time, by right clicking on an individual test case class and then continuing as before. The test classes are Robotium test cases, but be aware that the test cases have been tested only for the standard AVD described below.

As you implement various steps of the Lab, run the test cases every so often to see if you are making progress toward completion of the Lab.

Warnings:

1. We have done our testing on an emulator using a Galaxy Nexus 5 AVD with API level 26. To limit configuration problems, you should test your app against a similar AVD. Also, when testing, make sure that your device is in Portrait mode with the screen unlocked when the test cases start running.
2. The ImplicitTest test case requires that you've installed both the Lab2_Intents and the Lab2_MyBrowser applications. Remember - If you change Lab2_MyBrowser, you'll need to reinstall it.
3. Spelling counts. Pay attention to how to specify data values in your xml files. Leave out or misspell a single letter and Android won't understand what you really meant.

As you implement various steps of the lab, run the test cases every so often to see if you are making progress toward completion of the lab.

Once you've passed all the test cases, follow the following instructions to submit your work via GitLab for grading.

Submission

To submit your work you will need to commit your solution to your repo on GitLab by running the following command: git push origin master. Note: if you have not already pushed this branch to your repo on GitLab you will need to make a slight modification for this first time and run this instead: git push –u origin maste. This sets up tracking between your local branch and a branch with the same name on your repo in GitLab.

Challenge Activities

Exploring Different Intents

Problem Description:

In today's Intents Lab you used an Intent to launch a web browser. However, as you've seen in the lectures, Intents can be used to perform many other actions like opening the camera, placing a phone call or creating a new email.

Your goal in this challenge problem is to familiarize yourself with some other common Intents and add 3 buttons in your app's main activity which use Intents to do the following:

1.   Open the calendar to create a new calendar event.
2.   Open the map to show a specific location.
3.   Compose an email.

Note: Each of the these can be implemented using the Intents documented at this link: https://developer.android.com/guide/components/intents-common.html. We've chosen these Intents because they're available on your emulator (the Galaxy Nexus API 26). If you're not using this emulator and are receiving errors, it is most likely an issue with the device type or API version.

Once you have the buttons working, think about what the code of the receiving application might look like. What intent filters might they need to receive these intents?

Firing Intents with ADB

Problem Description:

The Android SDK comes with a command line tool called the Android Debug Bridge (adb). adb allows you to interact with your emulator or device via the command line.

To see for yourself, start your emulator and go to the Terminal. You can access the Terminal from inside Android Studio by going to View > Tool Windows > Terminal.
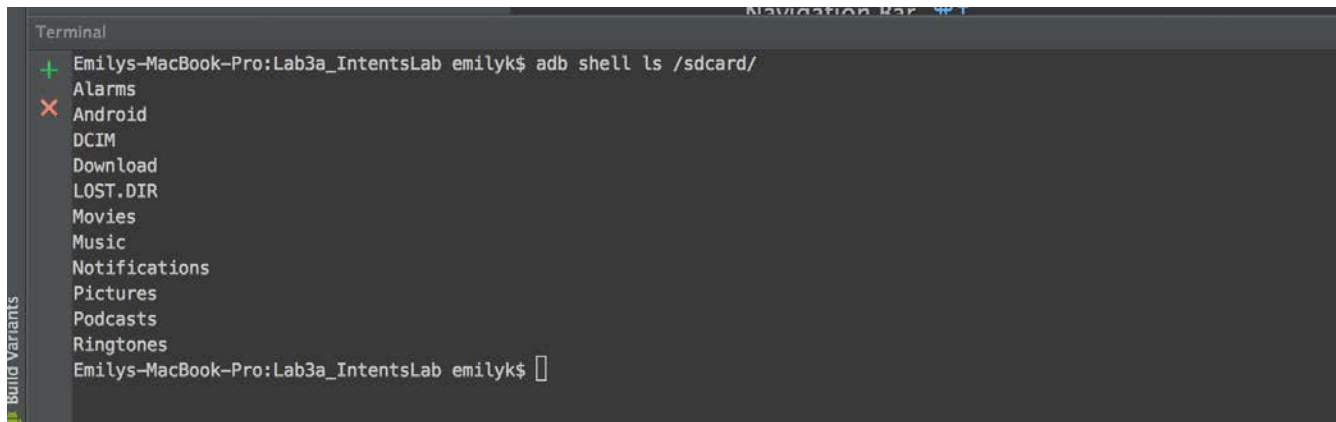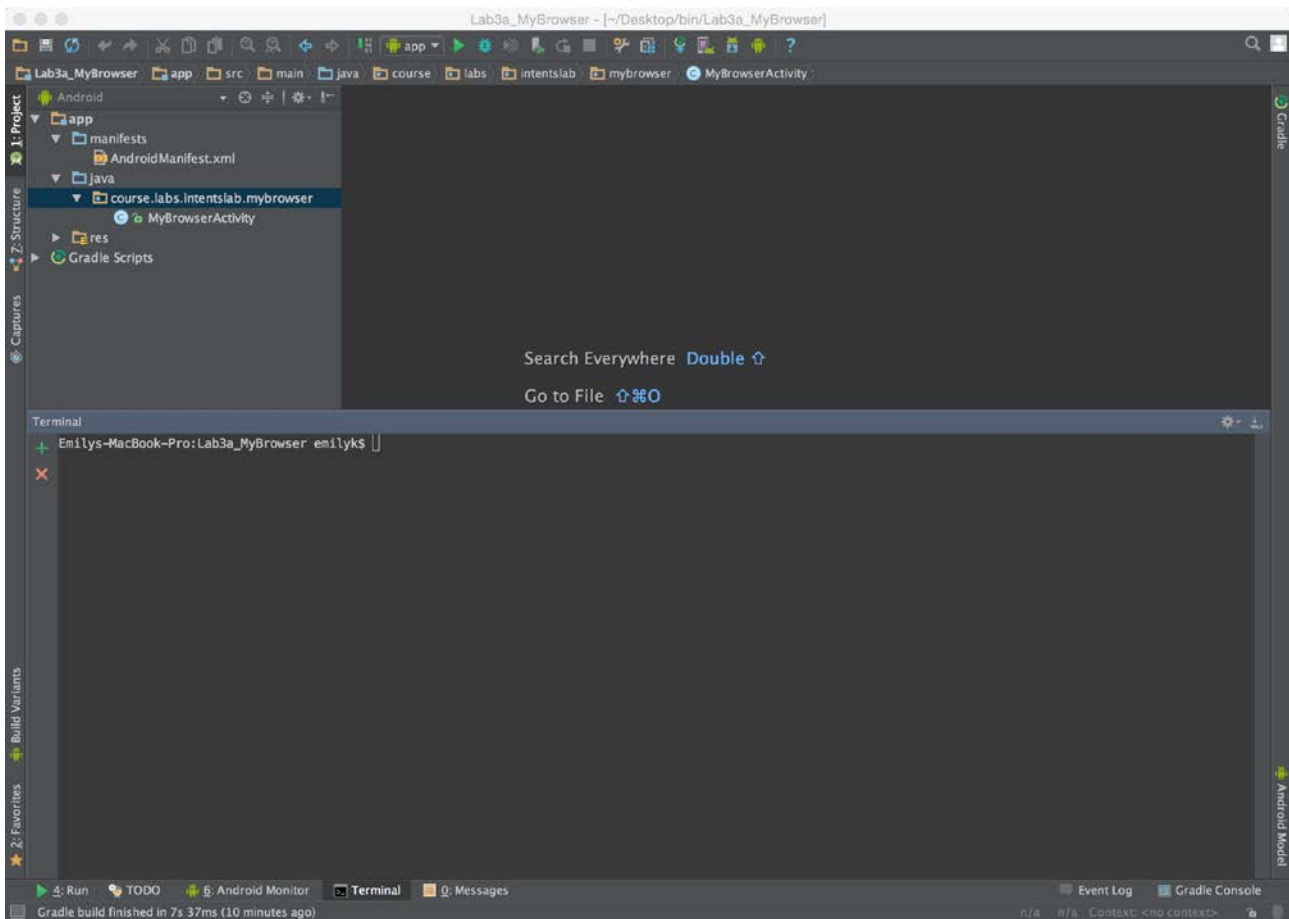
Once your emulator has finished booting, send the command 'adb shell ls /sdcard/' from the terminal.
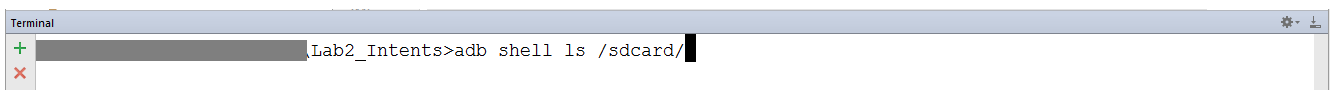
Note: If you receive an error saying the 'adb' command is not found, you most likely need to add the Android SDK to your path or use the full path to adb when calling. If you're not sure where it's installed, you can find this from within Android Studio by going to File > Project Structure and finding your Android SDK location (screenshot below). adb is located at <path_to_android_sdk>/platform-tools.
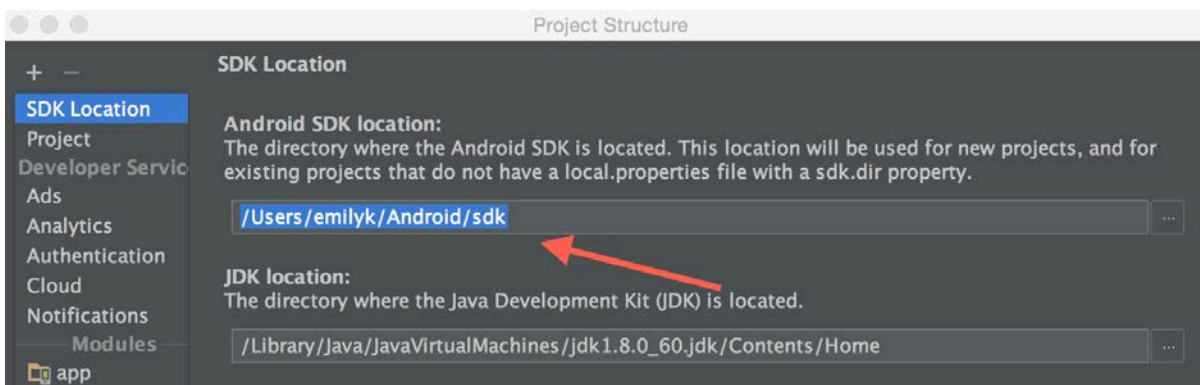
With this adb command, you've just instructed the emulator to run a shell command, 'ls /sdcard/'. If all goes smoothly, you should see something similar to the output below. This is the output from the 'ls' command and are the files and directories on your emulator's /sdcard.

adb has many features, but for today we will only focus on using it to fire Intents via the command line. This feature can be helpful to quickly verify that your app's Activity is responding correctly to Intents.

In general, we can fire an Intent via adb with the following command:

adb shell am start -a <ACTION> -c <CATEGORY> -t <MIME_TYPE> -d <DATA>
        -e <EXTRA_NAME> <EXTRA_VALUE> -n <ACTIVITY>

Most of these options should look familiar to you, and their values can be found in an app's AndroidManifest file. For example, suppose we want to open a map to a specific location using the MapsActivity in the com.google.android.apps.maps package. The map's AndroidManifest.xml file might look something like the below:

<manifest package="com.google.android.apps.maps" />
    …
    <activity android:name="com.google.android.maps.MapsActivity">
      <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:scheme="geo" />
      </intent-filter>
    </activity>
    …
</manifest>

Using the information in this file and providing the coordinates we'd like the map to open to, we can construct the following command to launch the Activity via an Intent:

adb shell am start -a android.intent.action.VIEW -d geo:38.98,-76.9 -n com.goo
gle.android.apps.maps/com.google.android.maps.MapsActivity

Note: If you are using a Nexus Galaxy API 26 emulator, go ahead and try this command. It should open Android's default maps app centered at the coordinates you've given it.

Your goal in this challenge problem is to use adb to open the MyBrowserActivity in the MyBrowser app you coded in the lab.

Hint: Use information in your MyBrowser's AndroidManifest.xml to construct the command.

Note: In order to test this, you'll want to make sure the MyBrowser app is installed on your emulator, but not currently open.

For more information on using ADB to fire intents:
- Verifying Intents via ADB: https://developer.android.com/guide/components/intents-com-mon.html#AdbIntents