CMSC436: Fall 2013 – Week 3 Lab

Objectives:

Familiarize yourself with the Activity class, the Activity lifecycle, and the Android reconfiguration process. Create and monitor a simple application to observe multiple Activity's as they move through their lifecycle. Once you've completed this lab you should understand: the Activity class, the Activity lifecycle, how to start Activity's programmatically, and how to handle Activity reconfiguration.

Overview:

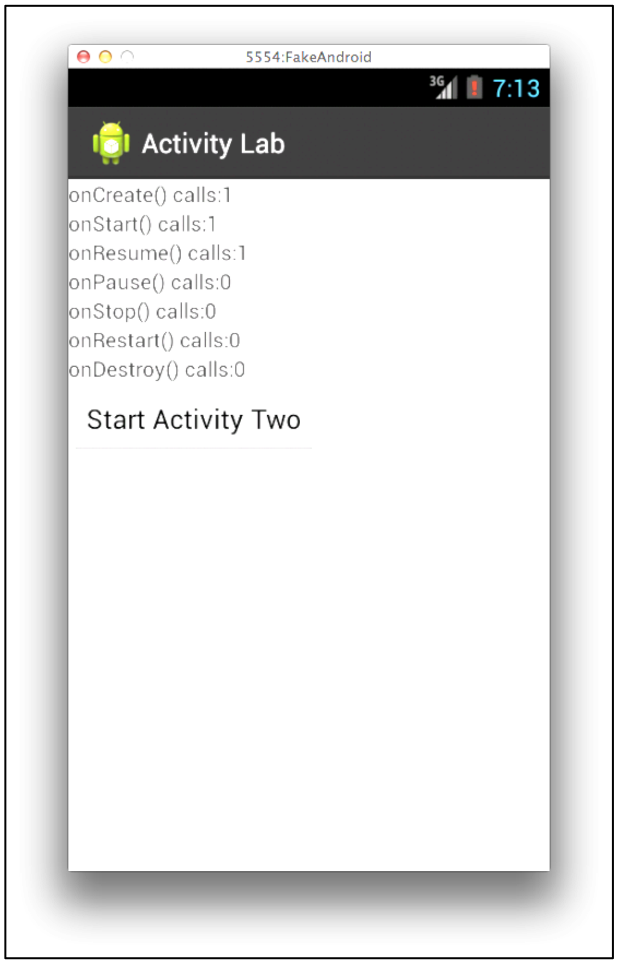
This lab has two parts. One which focuses on the Activity class, and one that focuses on the Intent class.

Part 1: The Activity Class

The goals of this exercise are 1) to familiarize yourself with the Android Activity Lifecycle and 2) to better understand how Android handles reconfiguration in conjunction with the activity lifecycle.

Exercise A:

The application you will use in this exercise will display a user interface like that shown below. We will provide the layout resources for this application, so you will not need to implement this.

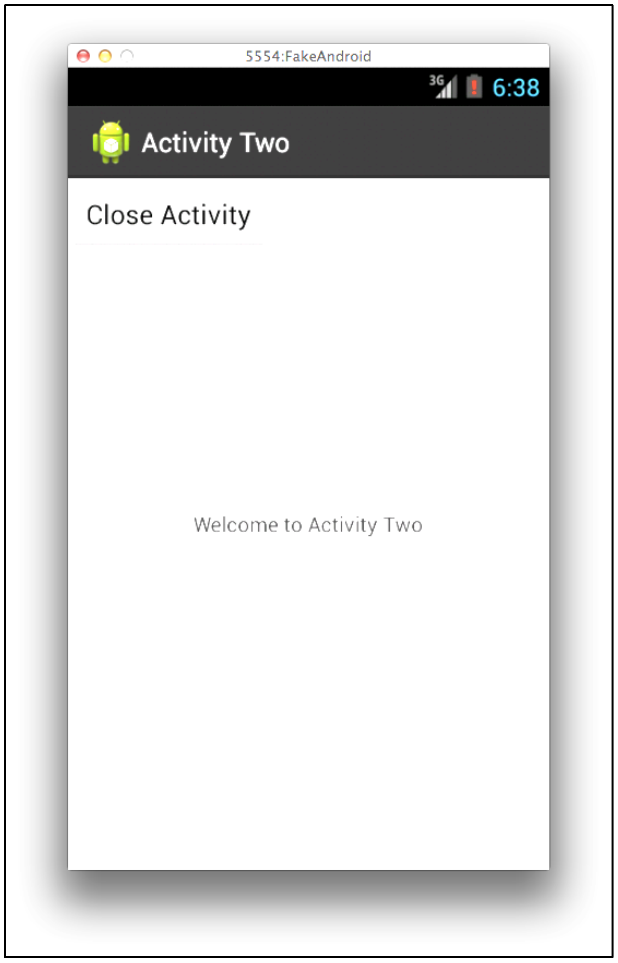


```
onCreate() calls:1
onStart() calls:1
onResume() calls:1
onPause() calls:0
onStop() calls:0
onRestart() calls:0
onDestroy() calls:0
```

This application comprises two activities. The first Activity, called Activity One will monitor the Activity class' lifecycle callbacks (onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy(), and onRestart()). The activity will output a Log message, using the Log.i() method, every time one of these lifecycle methods is called. It will also maintain one counter for each method, which counts the number of times that method has been called since the application started up.

The methods are their current invocation counts should always be displayed in Activity One's user interface.

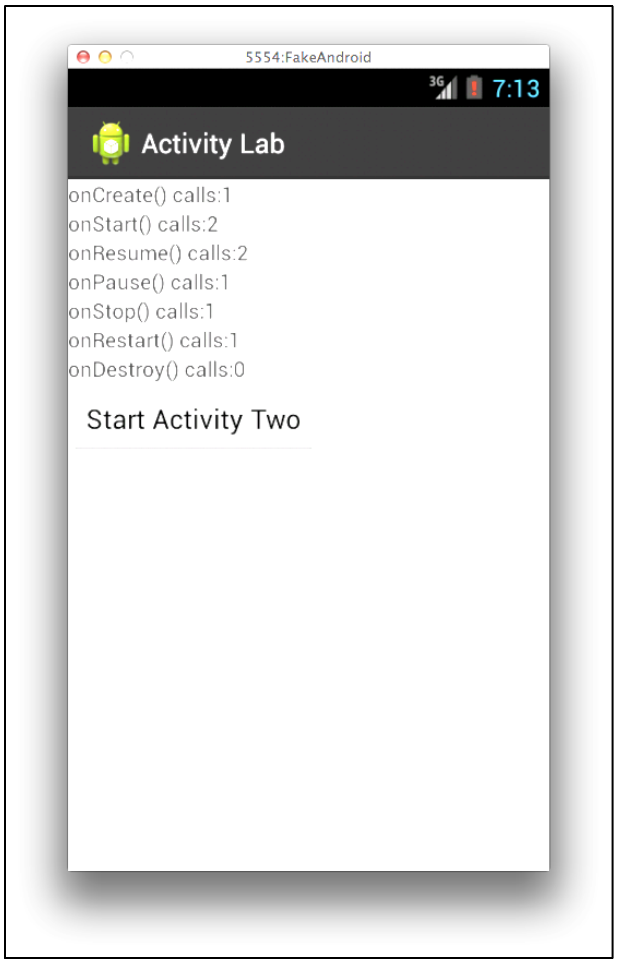
When the user clicks on the button labeled "Start Activity Two", Activity One will activate a second Activity, called Activity Two. As the user navigates between Activity One and Activity Two, various lifecycle callback methods will be invoked and their associated counters will be incremented. These counters should maintain the correct counts.



Welcome to Activity Two

Activity Two will display a text message and a button, labeled "Close Activity" to close the activity (the user may also press the Android Back Button to navigate out of the Activity). Again, we will provide you with the associated layout files, so you don't need to implement them. Just like Activity One, Activity Two will also monitor its Activity lifecycle callbacks and output a log message each time Activity Two executes one of the lifecycle callback methods.

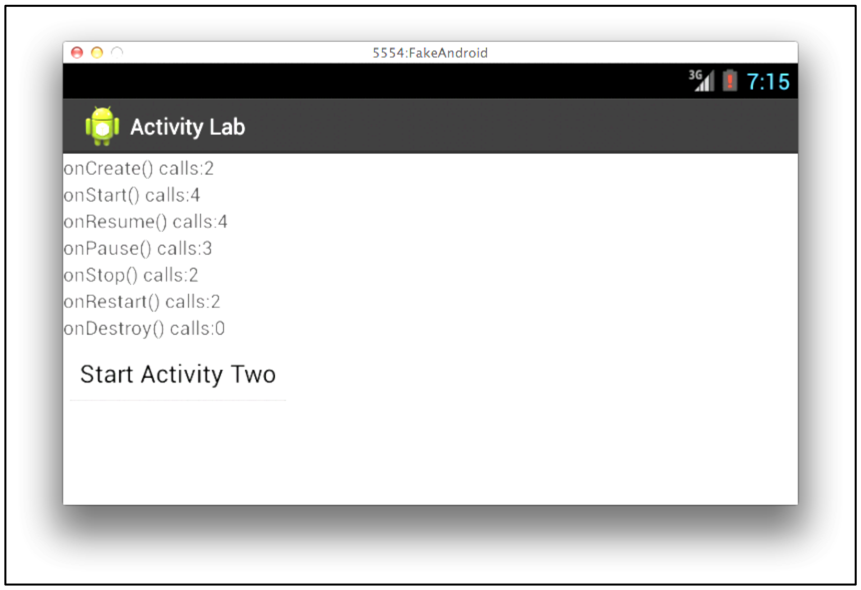
When the user navigates away from Activity Two and eventually returns to Activity one, make sure that Activity One's user interface displays the correct method invocation counts.



```
onCreate() calls:1
onStart() calls:2
onResume() calls:2
onPause() calls:1
onStop() calls:1
onRestart() calls:1
onDestroy() calls:0
```

Exercise B:

When a user reorients their Android device, changing, say, from Portrait mode to Landscape mode, or vice versa, Android, will normally kill the current Activity and then restart it. You can reorient you device in the emulator by pressing Ctrl+F12 (Command+F12 on Mac). When this happens and your current Activity is killed and restarted, the Activity's lifecycle callback methods will necessarily be called.



```
onCreate() calls:2
onStart() calls:4
onResume() calls:4
onPause() calls:3
onStop() calls:2
onRestart() calls:2
onDestroy() calls:0
```

In this exercise, you should modify your application from Exercise A so that the lifecycle callback invocation counters maintain their correct values even though the underlying Activity's are being killed and recreated.

To do this you will need to store, retrieve and reset the various counters as the application is being reconfigured. To do this you will need to save the counts in a Bundle as the Activity is being torn down, and you will need to retrieve and restore the counts from a Bundle as the Activity is being recreated. See "Recreating an Activity" at http://developer.android.com/training/basics/activity-lifecycle/for more information on storing and retrieving data with a Bundle.

Implementation Notes:

1. Download the application skeleton files from the Lectures & Labs web page and then import it into your IDE.

2. For exercise 1, implement the following classes and methods in ActivityOne.java.

    a. Create 7 counter variables, each corresponding to a different one of the lifecycle callback methods. You will need to increment these variables' values when their corresponding lifecycle methods get called.
    b. There are 7 TextView variables, which display the values of the count variables on the Activity One display. If you open layout.xml and click on each textview, you will see its id. These varibales are mainly used in the displayCounts() method.
    c. Override all the lifecycle callback methods. In each method, update the appropriate counter and call the displayCounts() to update the view on the UI.
    d. Implement the displayCounts () method.

    ```
    // Updates the displayed counters
    // Hint: Access the TextViews by calling Activity's findViewById().
    // Update the TextView's text by calling TextView's setText() method.
    // TextView textView1 = (TextView) findViewById(R.id.textView1);
    // textView1.setText("foo");

    public void displayCounts() {
    ...
    }
    ```

    e. Implement the launchActivityTwo() method.

    ```
    // This function launches Activity Two.
    // Hint: use Context's startActivity() method.

    public void launchActivityTwo () {
    ...
    }
    ```

3. For Exercise 2, implement the following extensions to the work you did in Exercise 1. Hint: See "Recreating an Activity" at http://developer.android.com/training/basics/activity-lifecycle/ for information on storing and retrieving data with a Bundle

    a. Implement the source code needed to save the values of the lifecycle callback invocation counters. When an Activity is being killed, Android calls onSaveInstanceState() . This gives the Activity a chance to save any per-instance data it may need later, should the activity be restored.

    ```
    // Save per-instance data to a Bundle (a collection of key-value pairs).
    public void onSaveInstanceState(Bundle savedInstanceState) {
    ...
    }
    ```

    b. Implement the source code needed to restore the values of the lifecycle callback invocation counters. There are different ways to do this. One way is to implement the restore logic in the onCreate() method.

    ```
    protected void onCreate (Bundle savedInstanceState)
       super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_one);

      // Do we have saved state?
      if (savedInstanceState != null){
          // Restore value of counters from saved state
      }
    }
    ```

    Another way to do this would be to override the onRestoreInstanceState() method. Be sure you understand when and why this method is called.

    ```
    protected void onRestoreInstanceState (Bundle savedInstanceState) {
       // Restore value of counters from saved state
    }
    ```

Deliverables:

Submit your project via the submit server. Your submission should include:

a. A zip file containing the source code that implements both Exercise 1 and Exercise 2.

b. A text file called Activity.txt with the answers to the following questions:

Steps:

1) Start up your application on your device or in the emulator.

2) Press the button to start Activity Two. Once Activity Two appears:

a) List the lifecycle methods have been invoked on Activity One, since the application started, in the order they occurred.

b) List the lifecycle methods have been called by Activity Two, since the application started, in the order they occurred.

3) Navigate back to Activity One, by selecting the "Close Activity" Button. Once Activity One appears:

   a) List the lifecycle methods have been invoked on Activity One, since the application started, in the order they occurred.

   b) List the lifecycle methods have been called by Activity Two, since the application started, in the order they occurred.

4) Press the button to start Activity Two. Once Activity Two appears, press the home button on your device (or the Home key for emulators, the home key is "fn + left arrow" on Macs)

   a) List the lifecycle methods have been invoked on Activity One, since the application started, in the order they occurred.

   b) List the lifecycle methods have been called by Activity Two, since the application started, in the order they occurred.

5) Start the application again. Once Activity One appears,

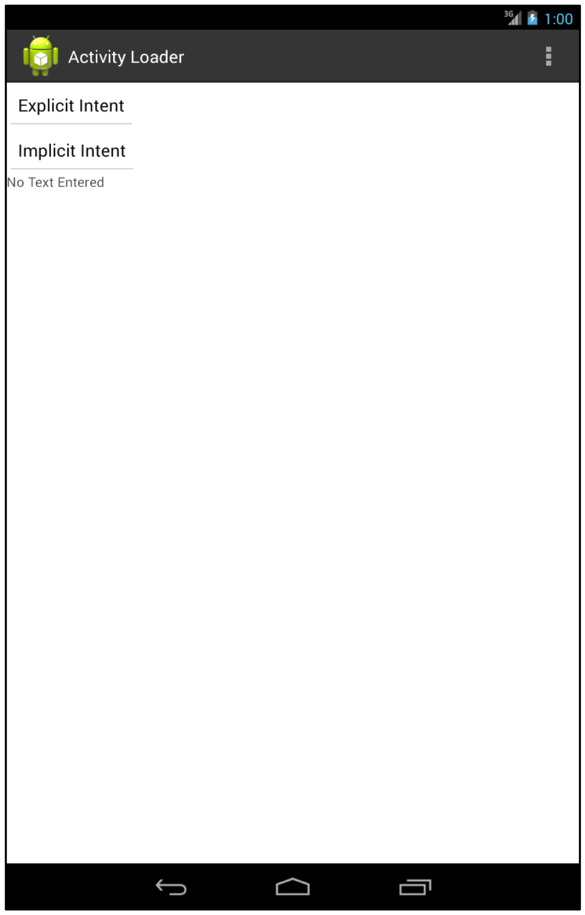
   a) List the lifecycle methods have been invoked on Activity One, since this time the application started, in the order they occurred.

   b) List the lifecycle methods have been called by Activity Two, since this time the application started, in the order they occurred.

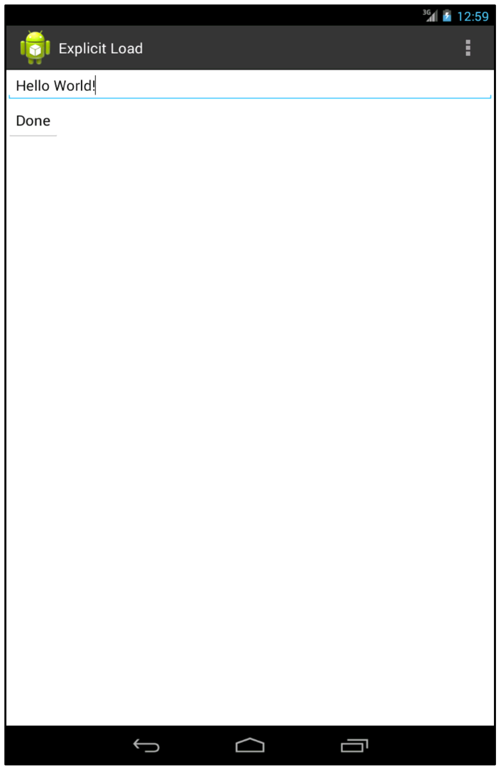
Part2: The Intent class

For this part of the lab you will use Intents to explicitly and implicitly activate Activities. In some cases, you will also receive results back from Activities that were started via the startActivityForResult() method. See http://developer.android.com/training/basics/intents/ for more information.

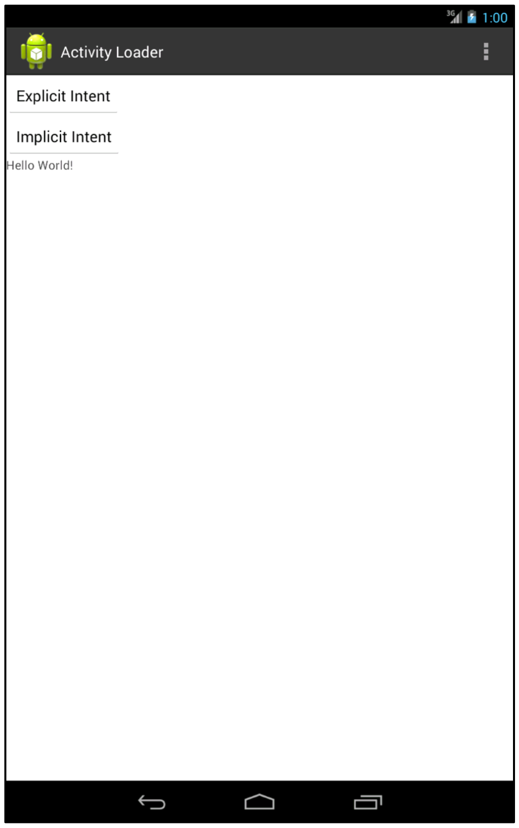
The main Activity, called "Activity Loader" should display two Buttons and one TextView.



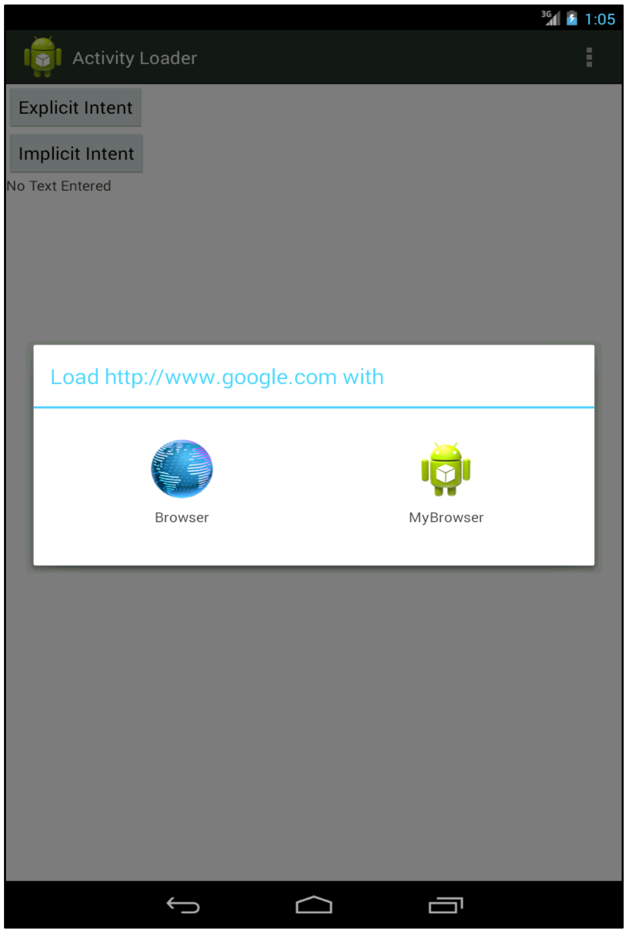
Clicking the "Explicit Intent" Button should launch a new Activity called "Explicit Load" that contains a large EditText and a Button. The user can enter text in the EditText View.



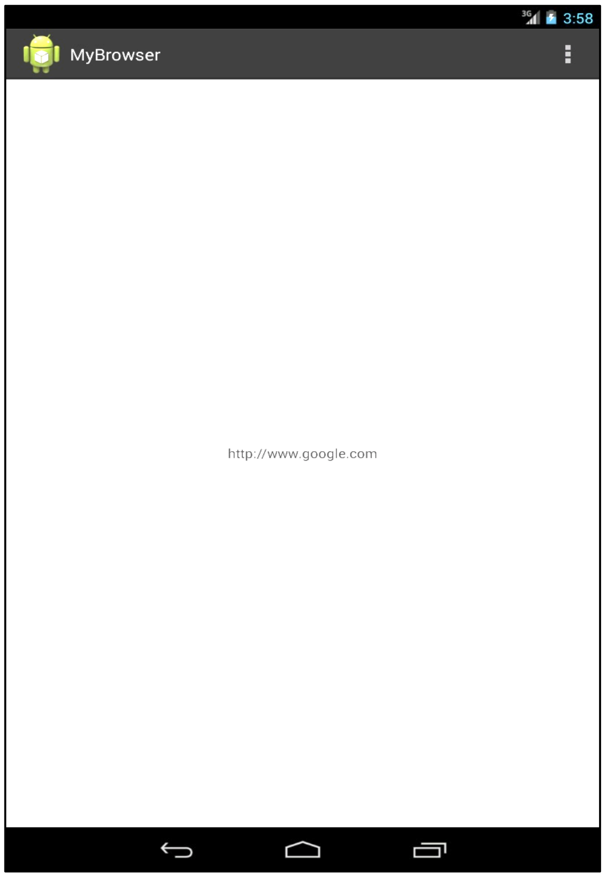
Clicking the Done button should cause the application to return back to the main Activity. Any text that was entered in the EditText view in the "Explicit Load" Activity should now appear in the TextView.



When the user clicks on the “Implicit Intent” button, the Activity Loader should create and use an Intent to implicitly activate an Activity to handle the URL “http://www.google.com”. Because multiple applications can handle such Intents, an App Chooser should appear, allowing you to select a specific application to handle the Intent. In particular, the Chooser should display at least two choices: the built-in Browser Activity, or your own MyBrowser Activity.



If the user selects the Browser Activity, it will open the URL and display the webpage at the given URL. If the user instead selects the MyBrowser Activity, it should simply display the text of the URL in a TextView.



http://www.google.com

Implementation Notes:

1) Implement launching the Explicit Load Activity and returning the text result to the main Activity. The Explicit Load Activity can be launched using a simple Explicit Intent, just as you used in Part 1. To return the text from the Explicit Load Activity, you can use the Intent.putExtra() method to store the text in the Intent to be returned, and then use Intent.getStringExtra() to read it back out. See the "Getting a Result from an Activity" section on the Android developer site (http://developer.android.com/training/basics/intents/result.html) for more information.

2) Implement launching an Activity to view a webpage. When the Implicit intent button is clicked you should create and start an Implicit Intent indicating that a webpage needs to be view. The system will automatically locate Activities that are capable of performing this Intent. See the "Sending the User to Another App" section on the Android developer site (http://developer.android.com/training/basics/intents/sending.html). Depending on what other applications are on your device, you may or may not see a system-provided application chooser.

3) Implement and deploy your own MyBrowser activity, which displays the text URL. Start by manually launching the MyBrowser activity and have it just display a hardcoded text URL.

4) Add the appropriate Intent Filters to MyBrowser so that the user will have the option to launch the application when the Activity Loader tries to start an Activity to handle its Intent. To indicate that the Activity can handle this type of Intent, you will need to add an <intent-filter> to the <activity>

in AndroidManifest.xml. Be sure to add the correct <action>, <category>, and <data> to the <intent-filter>.

5) Add code to acquire the URL from the Intent that starts the application. Use information in this Intent when you display the text URL.

You can find a skeleton for this Part on the Lectures & Labs web page. Below are the list of classes and functions you need to implement:

1. ActivityLoader.java

public void startExplicitLoad(View view): start the ExplicitLoad Activity.

public void startImplicitLoad(View view): create an implicit intent and the App Chooser (Hint: use Intent.createChooser) allowing you to select the built-in Browser Activity or your own MyBrowser Activity

protected void onActivityResult(**int** requestCode, **int** resultCode, Intent data): get the data that is returned from the ExplicitLoad activity and display it in the TextView.

2. ExplicitLoad.java

public void doneClicked(View view): get text from the EditText View, store it in a intent, and return it to the main Activity.

3. MyBrowser.java

protected void onCreate(Bundle savedInstanceState): Get the text URL from Intent and display it in the TextView whose id is “url”.

Deliverables:

- A zip file containing the source code