CS 234/334 Lab 1: Android Jump Start

Distributed: January 7, 2014

Due: Friday, January 10 or Monday, January 13 (in-person check off in Mobile Lab, Ry 167). No late assignments.

Introduction

The goal of this lab is to learn the fundamentals of developing Android Applications, from setting up development environment, building some applications that get data from device sensors, testing them on emulator, to installation on a physical device.

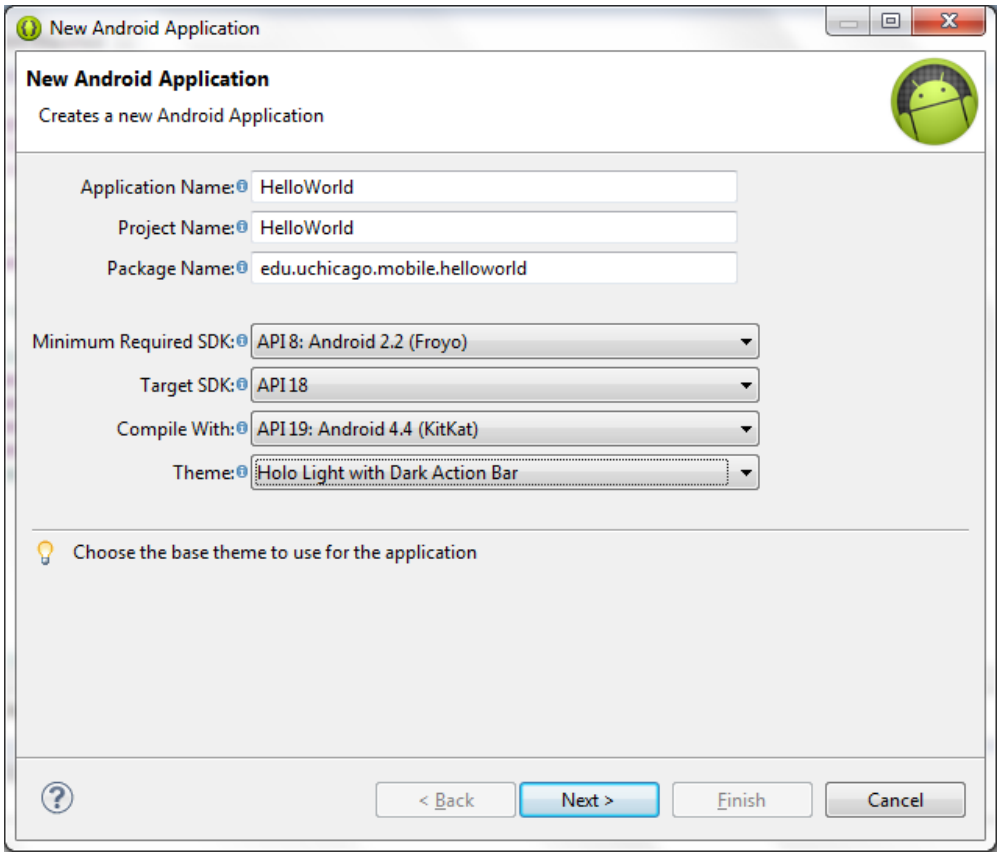
Objective

- Setup the development environment
- Create a "Hello world" application
- Understand the various parts of an Android project
- Use the Android Emulator
- Install and run the application on a physical device
- Create a simple User Interface application
- Create application that access internal sensors of a device

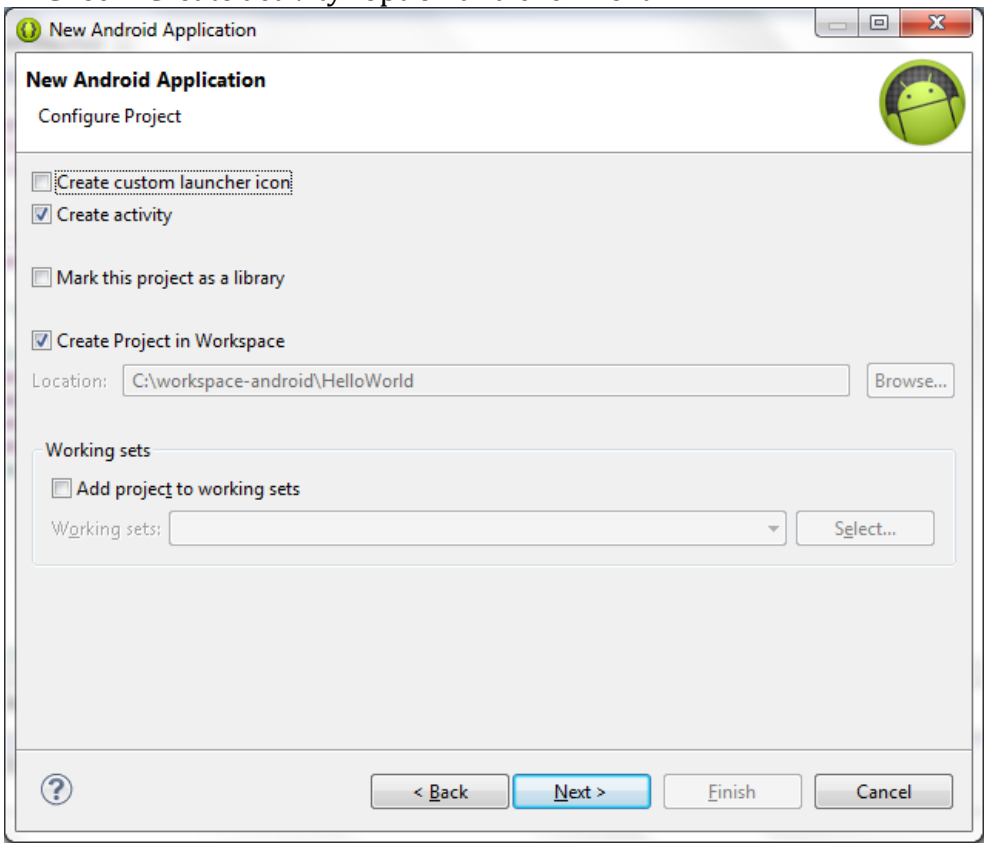
Instructions

1. Setup the development environment
   - We will use Eclipse as a main IDE for our work but before installing Eclipse we need to install Java first. The Java that we use is version 7. http://www.oracle.com/technetwork/java/javase/downloads/index.html
   - There are two ways to make Eclipse able to develop Android applications
     - Install ADT bundle, this is Eclipse with ADT plugin. Follow the instructions for ADT bundle installation on http://developer.android.com/sdk/installing/bundle.html
     - Separately install Eclipse and ADT plugin. Follow the instructions for this way on http://developer.android.com/sdk/installing/index.html

2. Create "Hello world" application
   - Open Eclipse
   - Click the menu **File -> New -> Project**
   - Expand the Android folder and select Android Application Project
     - Name the application "HelloWorld". Eclipse will automatically name the project and package name.
       - Change the package name to "edu.uchicago.mobile.helloworld"
     - Choose minimum required SDK to be "API 8: Android 2.2 (Froyo)"
     - Choose target SDK to be "API 18"
     - Choose compile with to be "API 19: Android 4.4 (KitKat)"
     - Click Next



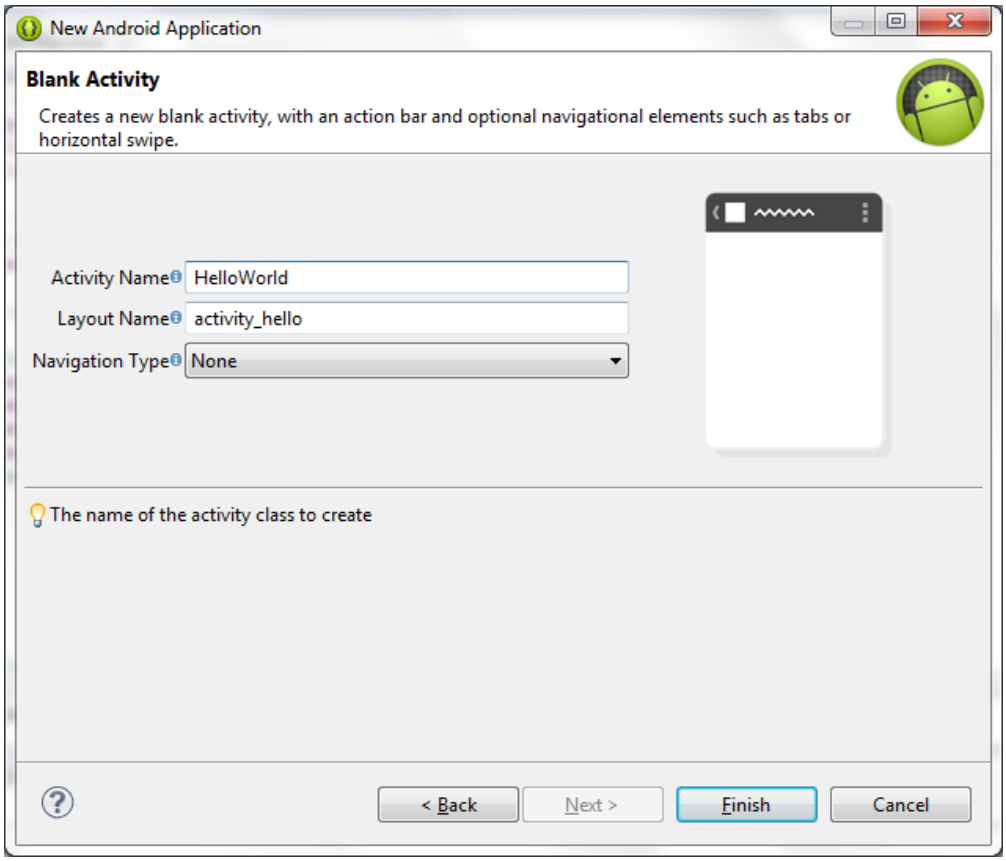
- Now we are in New Android Application page
  - Check "Create activity" option and click Next



- Now we are in Create Activity page
  - Check "Create activity"

- Choose “BlankActivity”
- Click Next

- In New Blank Activity page
  - Name activity to be “HelloWorld”
  - Name layout to be “activity_hello”
  - Set navigation to be None
  - Click Finish



3. Explore the project

The application you've just created is very similar to other java applications you may have created in Eclipse. Look in the PackageExplorer side bar. Notice that the Android Development Toolkit (ADT) has generated a number of folders and files for you:

- src: If you expand this out you'll see the package hierarchy you previously entered. This is where your source code files will go.
  - HelloWorld.java: This is the auto-generated stub Activity Class with the name you entered into the project creation wizard in Create Activity page. We will add some code to this later.
- res: This folder will contain all of the resources (a.k.a. external data files) that your application may need. There are three main types of resources that you will be using and the ADT has created a subdirectory for each.
  - drawable-*: This folder will hold image and animation files that you can use in you application. There are a lot of drawable directories. Android choose items from these directories depends on resolution of devices.
    - It already contains a file called ic_launcher.png which represents the icon that Android will use for your application once it is installed
  - layout: This folder will hold xml layout files that the application can use to construct user interfaces. You will learn more about this later, but using a layout

resource file is the preferred way to layout your UI.
    - It already contains a file called activity_hello.xml which defines the user interface for your "HelloWorld.java" Activity class. Double clicking on this file will open up the Android UI Editor that you can use to help generate the xml layout files.
- gen: This folder will contain Java files that get auto-generated by ADT. Notice that it already contains one file called "R.java"
    - R.java: This is a special static class that is used for referencing the data contained in your resource files. If you open this file you will see a number of static inner classes for each of the resource types, as well as static constant integers within them. Notice that the names of the member variables are the same as the names of the values in your resource files. Each value in a resource file is associated with an integer ID, and that ID is stored in a member variable of the same name, within a static class named after its data type.
        - The 'app_name' resource value has an ID and is of value type 'string'. The ADT automatically adds an integer constant to the R.string class and names it 'app_name'.
- assets: This folder is for asset files, which are quite similar to resources. The main difference being that anything stored in the 'assets' folder has to be accessed in the classic 'file' manipulation style. For instance, you would have to use the AssetManager class to open the file, read in a stream of bytes, and process the data. You will not be using assets quite as extensively as you will be using resources.
- AndroidManifest.xml: Every project has a file with this exact name in the root directory. It contains all the information about the application that Android will need to run it:
    - Package name used to identify the application.
    - List of Activities, Services, Broadcast Recievers, and Content Provider classes and all of their necessary information, including permissions.
    - System Permissions the application must define in order to make use of various system resources, like GPS.
    - Application defined permissions that other applications must have in order to interact with this application.
    - Application profiling information.
    - Libraries and API levels that the application will use.
- project.properties: This file contains all of the project settings, such as the build target you chose in the project creation wizard. If you open the file, you should see 'target=android-17', which is your build target. You should never edit this file manually. If you wish to edit the project properties, do so by right-clicking the project in the 'Package Explorer' panel, and selecting 'Properties'.

The project creation wizard has written the 'Hello World' application for you already. A string resource containing the display text has been placed into the res/values/strings.xml file. The value is named 'hello_world'.

The xml UI layout has been added to res/layout/main_activity.xml. While you can use the Android Layout Editor to create your xml layout, you can also code them manually yourself in XML by choosing tab "activity_hello.xml" next to tab Graphical Layout.

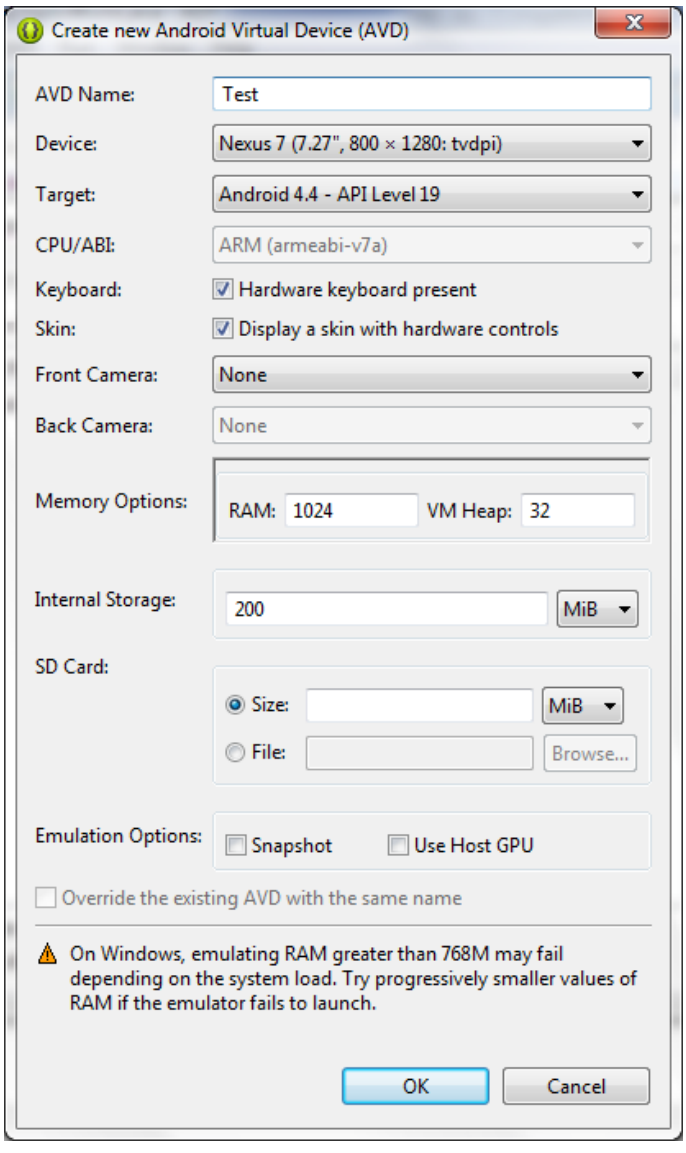
Notice there is TextView node under RelativeLayout node. Layout is used to define the layout of pages or some part of pages. TextView is UI element to display text. It has three properties: width, height, and the text to display. Notice that the text property is not set to "Hello World!". Instead it is set to reference the resource value which contains the text we

want to display. In this case we are choosing to display the contents of the 'hello' value. We do this by using the '@' symbol, followed by the value type of the resource (which is a 'string'), followed by the name of the value (which is 'hello').

4. Run our "Hello world" application on an emulator

You need to first create an Android Virtual Device (AVD). An AVD is a device configuration for the Android emulator that allows you to model different devices.

- Select the menu **Window -> Android Virtual Device Manager**
- Select Android Virtual Device tab
- Click New button
- Name an AVD "Test"
- Choose "Nexus 7" for device
- Choose "Android 4.4 – API Level 19" for target
- Click OK



Now we have Nexus 7 emulator that runs Android 4.4 for testing our application. The next step is running it

- Select our project in Package Explorer
- Select menu **Run -> Run**
- Select Android Application

- Click Run
  - Note: The emulator may take a long time to start up
  - Note: Another way to run your application is to right-click on the project in the Package Explorer, then select **Run As -> Android Application**.
  - You can interact with the emulator using the mouse just like you would with a device. To get started, press the Menu key to see the home screen.

5. Run our application on real device

Before we can run the application on a physical device we need to make a configuration change on the phone, and install some drivers for the phone on our development machine. We begin by making your project declare itself as debuggable. It's possible to do this through the Android Manifest Editor that the ADT provides, however doing this manually helps you understand the manifest better:

- Phone Modification
  - On most devices running Android 3.2 or older, you can find the option under **Settings -> Applications -> Development**.
  - On Android 4.0 and newer, it's in **Settings -> Developer** options.
  - On Android 4.2 and newer, Developer options is hidden by default. To make it available, go to **Settings -> About phone** and tap Build number seven times. Return to the previous screen to find Developer options.
- Installing the Android USB drivers
  - Mac OS X: Don't need to install drivers, it should just work.
  - Windows: Follow instructions here, http://developer.android.com/sdk/win-usb.html
    - Even if the device is recognized by Windows, you may need to install from OEM USB Drivers -> Asus -> Nexus 7 in order to deploy applications to the device. From the Device Manager, find the unknown device Nexus 7 and point it to the directory where you extracted the driver from Asus.
  - Linux: Follow instructions here, http://developer.android.com/tools/device.html#setting-up

Ensure the device is properly connected. Run the application as you would normally. The “Hello World!” app should start on the phone.

6. Build simple application

There are four major types of component classes in any Android application:

- Activities: Much like a Form for a web page, activities display a user interface for the purpose of performing a single task. An example of an Activity class would be one which displays a Login Screen to the user.
- Services: These differ from Activities in that they have no user interface. Services run in the background to perform some sort of task. An example of a Service class would be one which fetches your email from a web server.
- Broadcast Receivers: The sole purpose of components of this type is to receive and react to broadcast announcements which are either initiated by system code or other applications. If you've ever done any work with Java Swing, you can think of these like Event Handlers. For example, a broadcast announcement may be made to signal that a WiFi connection has been established. A Broadcast Receiver for an email application listening for that broadcast may then trigger a Service to fetch your email.
- Content Providers: Components of this type function to provide data from their application to other applications. Components of this type would allow an email

application to use the phone's existing contact list application for looking up and retrieving an email address.

In this lab, we will be focusing on what Activities are and how they are used. We will cover the other components in later labs. If you would like more information on these components, visit the Android overview page for Application Components, http://developer.android.com/guide/components/fundamentals.html#Components

In case you haven't figured it out by now, you have already created one of these component classes. That's right, the HelloWorld class is an Activity Class. It's a simple user interface designed to greet the user. In the section that follows, we'll make our application more personal by adding a new Activity class to ask for the user's name. We'll then update the existing HelloWorld greeting Activity to display that name.

1. *Getting the user's name*

To get the user's name, you will be creating an Activity class which will allow the user to enter their name into a text field and press a button when finished to proceed to the HelloWorld greeting Activity. There are three separate steps to accomplish here. You must first layout your user interface in XML. Then you must create the Activity class to parse the input from the user and initiate the HelloWorld Activity. Finally, you will have to reconfigure the application to use your new name retrieval Activity on startup.

a) Create the User Interface

Android allows you to layout your user interfaces using a simple XML specification. We will go into more depth on this topic in the next lab, so for now you will be setting up a basic interface using four different GUI elements. Begin by creating a new Android XML file:

- Select the menu **File -> New -> Android XML File**.
- Ensure the Project matches the name of your project and that the resource type is Layout.
- Enter "name_getter.xml" as the file name
  - The name of your layout files must only contain lower case letters, the numbers 0-9, underscores '_', or periods '.'
- Select LinearLayout for Root Element
- Click Finish. The file will be in /res/layout.
- By default, the file will be opened to the Layout Editor tab. Select the tab labeled name_getter.xml to switch to the XML Editor.

Each GUI element derives from the View base class. The first element was added for you when you created the XML layout file and selected LinearLayout from the drop-down menu. You should be able to see an XML opening and closing tag labeled LinearLayout in the editor. Each XML layout file must have a single root view tag, inside which all other view tags are nested. The LinearLayout tag tells Android to arrange elements contained inside it in a straight line in the order in which they appear. Let's make a few modifications to the LinearLayout by editing the attributes contained in the opening LinearLayout tag:

- Ensure the attributes labeled android:layout_width and android:layout_height are set to "match_parent" (Include the quotes).
  - This tells Android that the LinearLayout should take up all the available width and height on the screen.
- Add an attribute labeled android:orientation and set it to "vertical", if there is no such attribute.
  - This tells Android that elements nested inside the LinearLayout should be laid out in a column, as opposed to a single row as indicated by "horizontal".

Let's add the other three UI elements to our XML layout file:

- Switch back to the Layout Editor tab.
- Expand the folder icon labeled “Form Widgets”
  - Click and drag the TextView onto the white canvas.
  - The Layout Editor will pre-populate the label with its auto-generated id, which may look somewhat strange.
- Expand the folder icon labeled “Text Fields”
  - Click and drag the PlainText onto the white canvas (may be simply called “abc”).
  - Remember, order matters for the LinearLayout.
- Expand the folder icon labeled “Form Widgets” again
  - Click and drag the Button onto the white canvas.

This is what you want your UI to look like. However it may not resemble this quite yet:

- Switch back to the XML Editor to change the attributes of the elements you just added.
  - Notice that all the UI elements you added are nested within the LinearLayout Element.
  - There will always be only one root element. You may, however, nest other Layout elements within each other.
- Editing the TextView element:
  - This is the label that prompts the user to enter their name. It displays the value contained in the android:text attribute.
    - Set android:text attribute to ask a user to enter their name, android:text=“What's your name”
  - The android:id attribute provides a variable name for referencing this element from within the code.
    - Id's are specified with the syntax of @+id/MyId01 .
    - MyId01 is the handle used to identify the element from within your application code via the Static R class.
- Editing the EditText element:
  - This is the text field where the user will input their name. It will default to contain the value set by the android:text attribute.
    - Remove this attribute, if it exists, we don't need it.
  - The android:hint attribute provides a hint to the user when the field is empty, and disappears when text is entered.
    - Set this attribute to instruct the user to enter their name, android:hint=“Enter your name”. You will need to add the attribute if it doesn’t exist.
  - Either make a mental note of the android:id attribute or provide your own variable name which you will use to reference this element from within the code. Assume we name it MyId02.
- Editing the Button element:
  - This is the button that will allow the user to continue to the next HelloWorld greeting screen.
  - It displays the value contained in the android:text attribute.
    - Set this attribute to something like "ok", "next", or "submit".
  - Either make a mental note of the current value for the android:id attribute or provide your own variable name which you will use to reference this element from within the code. Assume we name it

MyId03.

b) Create the Activity Class

Using the HelloWorld Class from section 2 as an example, we will create new activity.

- Create new class.
  - Select menu **File** -> **New** -> **Class**.
  - Name it Greeting.
  - Make it be under our package, edu.uchicago.mobile.helloworld
  - Make it extend android.app.Activity and implement android.view.View.OnClickListener interface.
  - Click Finish.
- If not automatically created, implement the OnClickListener interface by creating a method stub with the following signature: public void onClick(View v).
  - We'll fill in this method later
- Declare a member variable of the type android.widget.EditText
  - This will hold a reference to the text field in which the user will enter their name, the same one that you added to the name_getter.xml layout file.
- Add a method with the following signature: public void onCreate(Bundle savedInstanceState) .
  - This method will be called when the Activity starts and is where initialization of local and member data will be done.
- Inside this method perform the following:
  - Make a call to super.onCreate(savedInstanceState)
    - This should always be done and is to ensure that any necessary parent class initializations are performed.
  - Make a call to this.setContentView(R.layout.name_getter)
    - When you created the XML layout file earlier, the Android Eclipse Plugin automatically added a static constant to the static R.layout class in the R.java file under the /gen folder. This constant variable has the same name of the file and its value is used to identify the layout file.
    - This call tells Android to create a screen based off of the layout file.
  - Make a call to this.findViewById(R.id.<EditText id>) and set your EditText member variable equal to the return value.
    - <EditText id> should be replaced with the android:id that was specified in the name_getter.xml layout file for the EditText element.
    - You will have to explicitly cast the return value to the type of EditText as this method only returns objects of type Object.
    - This static constant value was added in the same way as it was done for the R.layout.name_getter value and serves the same purpose.
    - The code will be like this

```
editText = (EditText) this.findViewById(
      R.id.MyId02);
```

  - Make a call to this.findViewById(R.id.<Button id>) and set a local android.widget.Button reference equal to the return value.
    - <Button id> should be replaced with the android:id that was specified in the name_getter.xml layout file for the Button element.

        - The code will be like this
```
Button button = (Button) this.findViewById(
      R.id.MyId03);
```
    - Make a call to button.setOnClickListener(this).
        - Button should be replaced with the local Button reference you just retrieved.
- Fill in the onClick method stub:
    - Retrieve the user entered text from the text field and keep it in a local String variable
    - Create an android.content.Intent object: new Intent(this, HelloWorld.class).
        - We'll use the Intent object to start the greeting activity and pass it information
        - If you want to study in depth in Intent, you should study this link, http://developer.android.com/training/basics/intents/index.html
        - The code will be like this
```
Intent intent = new Intent(this, HelloWorld.class);
```
    - You will use the intent.putExtra(<key>, <value>) method to pass the user entered name to the HelloWorld greeting Activity. This method functions like a hashmap, where values can be stored by associating them with a string key and later retrieved with the same key. You can retrieve this hashmap later by calling getExtras().
        - Make a call to <intent>.putExtra(<key>, <value>) ,
        - <intent> should be replaced with the intent object you just created.
        - <key> should be replaced with a string you will use to access the user's name later.
        - <value> should be replaced with the user's name obtained from the text field. To obtain the text from the EditText object, you must call to <editText object>.getText().toString() .
        - The code will be like this
```
intent.putExtra("name",
      editText.getText().toString())
```
    - Make a call to this.startActivity(<intent>)
        - This command will initiate the switch to the HelloWorld greeting Activity.

c) Reconfigure the HelloWorld Application

The Android Manifest contains information on all of the Application's components, including which component should be used in different scenarios. We need to tell our application to use the new activity on startup instead of the HelloWorld Activity which it was previously using. Begin by Double-Clicking the AndroidManifest.xml file to open it

- Like the Layout file there is also a Manifest Editor. Switch to the XML Editor by clicking the AndroidManifest.xml tab along the bottom of the editor
- Find the first opening <activity ... > tag and change the attribute labeled android:name from "edu.uchicago.mobile.helloworld.HelloWorld" to "edu.uchicago.mobile.helloworld.Greeting"
    - Look at the package attribute in the <manifest> tag, this declares the top level package for the application. If your activity resides in a sub-

package, then you must also include the sub-package in the name of your Activity class.
    - For example:
        - application package name in the manifest tag: android:package="my.app.basepackage"
    - The Intent Filter tag you see nested inside the Activity tag is what tells the application that this Activity is the Main, or startup, Activity.
        - Now Greeting Activity is our startup page.
- Add the following opening and closing Activity tag pair underneath the Activity tag pair you just modified, nested inside the Application tag:
    - <activity android:name="HelloWorld" ></activity>
    - This declares to the Android device that the application has an Activity component named HelloWorld. If you don't add this tag, Android will not let you launch this Activity

At this point, you should be able to try running the application. The application should start up and display the name retrieval activity you just created. You should be able to enter your name in the text field and hit the button, after which the old HelloWorld greeting activity should appear.

2. *Greeting the User*

Now that we've got the name retrieval activity completed, let's update the HelloWorld greeting Activity to print out the name the user entered. In the OnCreate method of HelloWorld.java:

- Make a call to this.getIntent().getExtras() to retrieve the hashmap in which you placed the user entered name.
    - This will return an android.os.Bundle object which acts like a hashmap.
    - You should check to ensure that this value is not null.
        - The code will be like this

```
Bundle extras = this.getIntent().getExtras();
if(extras != null) { //do anything in this block }
```

- Retrieve the user entered name from the Bundle object.
    - Make a call to the bundle's getString(<key>) method.
    - <key> should be replaced with the key String you used to put the user entered name into the hashmap in the name retrieval Activity's onClick method.
    - You should check to ensure that this value is not null.
        - The code will be like this

```
String name = extras.getString("name");
```

- Get a reference to the TextView object in your activity_hello.xml layout file (you may need to add an id field).
- Set the text of the TextView object to say Hello <name>!
    - <name> should be replaced with the user entered name that you retrieved from the bundle object.
- The code will be like this

```
TextView textView = (TextView) this.findViewById(
      R.id.text);
textView.setText("Hello " + name);
```

- Run your application.

7. Tracking you device location

In the following section, create new project name ShowingLocation with blank activity. At this point, you will get the simple Hello World application. We will modify the TextView to

show our location in Latitude and Longitude instead of "Hello world". The Android SDK does not currently provide a method for simulating GPS on the emulator, so the following section must be tested on a physical device to ensure that it is working properly. It would be helpful to look over the Android documentation on location, http://developer.android.com/guide/topics/location/index.html and http://developer.android.com/guide/topics/location/strategies.html

- Enable application to access location of the device
  - In AndroidManifest.xml add permission <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" /> under manifest node.
- Make our main activity to be LocationListener
  - Declare all of the methods of LocationListener interface.
    - `public void onLocationChanged(Location arg0)`
    - `public void onProviderDisabled(String arg0)`
    - `public void onProviderEnabled(String arg0)`
    - `public void onStatusChanged(String arg0, int arg1, Bundle arg2)`
  - Do not implement them now. Just declare them.
- Get LocationManager when the activity is created and request location update from Network and GPS as frequent as possible by passing the main activity as a LocationListener.
  - The code will be like this

```
LocationManager locationManager = (LocationManager)
      this.getSystemService(Context.LOCATION_SERVICE);
locationManager.requestLocationUpdates(
      LocationManager.NETWORK_PROVIDER, 0, 0, this);
locationManager.requestLocationUpdates(
      LocationManager.GPS_PROVIDER, 0, 0, this);
```

- Implement method onLocationChanged to show current location.
  - This method will be called when the device detects a change of the location.

8. Hand in

Show your greeting application and motion monitoring application to the CS234 TA (Connor) at the lab (Ryerson 167) on Friday January 10, from 1-3pm and 5–6pm or Monday January 13, 10am–12pm.