Tutorial #1

Android Application Development
Advanced Hello World App

1. Create a new Android Project

1. Open Eclipse
2. Click the menu **File** -> **New** -> **Other**.
3. Expand the **Android** folder and select **Android Application Project**.
4. Select to **Android Application Project** and then click "Next."
5. Name the project "**Hello**"
6. Change the package name. Select the SDK info you want. Click "Next."
7. Keep the default settings for the application project and click "Next."
8. Select "Blank Activity" and click "Next."
9. Use the default, "MainActivity" for the activity class name, install any dependencies needed and then press "Finish" to create the project.

2. A Tour of the Application

The application you've just created is very similar to other java applications you may have created in Eclipse. Look in the **Package Explorer** 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.

- **MainActivity.java**: This is the auto-generated stub Activity Class with the name you entered into the project creation wizard. We'll add some code to this later.
- **Android 4.x**: This is the version of the library you had chosen in the project creation wizard. The application will be built using this version of "android.jar."

- **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. *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 main.xml which defines the user interface for your 'Lab1.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.*

- **values**: This folder will hold files that contain value type resources, such as string and integer constants. *It already contains a file called strings.xml. Double clicking on this file will open up the Android Resource Editor. Notice that there are two strings in there already, one of which is named 'app_name'. If you select this value, on the right hand side of the editor you should see the Application Name you entered in the project creation wizard. You can use this editor to add new resources to your application.*

- **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'. A debugging hint: Occasionally, an Android project will report errors in Eclipse that do not show up in any source code file. Sometimes you can fix this by deleting R.java. When you rebuild your project, R.java gets generated, and perhaps your mysterious errors will disappear. A second hint: I encourage you to turn on build automatically in Eclipse (Project menu).

- **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 Receivers, 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-14," 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 "Lab1" 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."

  The xml UI layout has been added to res\layout\main.xml. While you can use the Android Layout Editor to create your xml layout, you can also code them manually yourself.

  Let's take a look at this file:

  - **Right-Click** on the file.
  - Select **Open With** -> **Text Editor**.

  Notice the Top Level node, **Linear Layout**, which defines the style of layout the file will be using. This "Linear Layout" is perhaps the most basic, and specifies that UI elements will be laid out in a continuous line.

  It has three properties: orientation, width, and height.

  Notice the Second Level node, **TextView**, which defines a UI element for displaying 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").

3. Run "Hello World" on the Emulator

Before we can run the application, we need to setup an Android Virtual Device (AVD), or emulator, to run it on:

- Select the menu **Window** -> "**AVD Manager**."
- Click the **New** button.
- Give your AVD a name.
- Select the target build that we would like to run the application on.
- Click **Create AVD** and close out the AVD Manager.

We're now ready to run our application.

- Select the menu **Run** -> **Run**

  *Note: The emulator may take a long time to start up. 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.

Congratulations! You've just created and an Android Application.

4. Simple Activity Classes

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.

In case you haven't figured it out by now, you have already created one of these component classes. That's right, the Lab1 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 Lab1 greeting Activity to display that name.

*Note: If you ever have problems getting things to compile in Eclipse, you might try **Project -> Clean**. You can also try to delete R.java under res, then **Project -> Build Project**.*

5. 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 "Hello" 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 Main Activity. Finally, you will have to reconfigure the application to use your new name retrieval Activity on startup.

6. 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 -> Other**.
- Expand the **Android** folder
- Select **Android XML Layout File** and click **Next**

Ensure the Project matches the name of your project and that the folder is **/res/layout**. *Layout files should always go in this folder.*

- Enter "**name_getter**" as the file name
- Select **LinearLayout** from the "Select the root element..." lower window pane
- Click **Finish**

- Open the new file in the Layout Editor tab. Select the tab labeled **name_getter.xml** to switch to the XML Editor. *This should be located in the bottom left corner of the Layout 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 selection window. 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:

- Set the attributes labeled: android:layout_width and android:layout_height to "fill_parent" (Include the quotes). This tells Android that the LinearLayout should take up all the available width and height on the screen.
- Leave the attribute labeled android:orientation set to "vertical." 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."

Now, let's add the other three UI elements to our XML layout file:

- Switch back to the "Graphic Layout" editor tab.

- The three elements we will add all reside under the category labeled "Form Widgets" and "Text Fields."

- Open the item category labeled **Form Widgets**

- Click and drag a **TextView** onto the canvas. The Layout Editor will pre-populate the label with its auto-generated id, which may look somewhat strange.

- Repeat the previous step for the **Text Field** and **Button** labels. For the text field item go into the "Text Fields" category and select "Plain Text." Remember, order matters for the LinearLayout. 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 this attribute to ask the user what their name is. 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.

  The information you are changing in this element looks like this:

  android:text=*"Enter your name:"* />

- Editing the **EditText** element:

  This is the text field where the user will input their name. 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. 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.

  The information you are changing in this element looks like this:

  android:hint = *"Enter your name"*

- Editing the **Button** element:

  This is the button that will allow the user to continue to the next application 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.

- Close and Save the name_getter.xml file.

7. Create a New Activity Class

- Create a new class that extends android.app.Activity class and implements android.view.View.OnClickListener interface. 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.

  To create the new class do the following:

  - Select "File" -> "New" -> "Other" -> "Android Activity"
  - Select a "blank Activity" and call the new class GetName
  - It should be in the source folder: Hello/src
  - Use the same package as before: edu.itu.Hello

  The new class will look like this:

```
package edu.itu.Hello;

public class GetName {

   }
```

Modify it so it extends Activity & implements android.view.View.OnClickListener: The finished class code looks like this so far:

```
package itu.edu.hello;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;

public class GetName extends Activity implements
android.view.View.OnClickListener{

     public void onClick(View arg0) {
          // TODO Auto-generated method stub

     }
}
```

- Declare a data member of type android.widget.EditText like this:

```
EditText name;
```

  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.

```
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);

```
public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           }
```

This should always be done and is to ensure that any necessary parent class initializations are performed. Also make a call to this.setContentView(R.layout.name_getter);

```
public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           this.setContentView(R.layout.name_getter);
      }
```

When you created the XML layout file earlier, the Android Eclipse Plug-in 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 it's 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.

```
public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           this.setContentView(R.layout.name_getter);

           name = (EditText) this.findViewById(R.id.editText1);
           }
```

- 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.

```
public class GetName extends Activity implements

android.view.View.OnClickListener{

     EditText name;
     Button button;

     public void onClick(View arg0) {
          // TODO Auto-generated method stub
     }
      public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           this.setContentView(R.layout.name_getter);

           name = (EditText) this.findViewById(R.id.editText1);
           button = (Button) this.findViewById(R.id.button1);
      }
     }
```

- Make a call to **button**.setOnClickListener(this); **button** should be replaced with the local Button reference you just retrieved.

```
public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           this.setContentView(R.layout.name_getter);

           name = (EditText) this.findViewById(R.id.editText1);
           button = (Button) this.findViewById(R.id.button1);

           button.setOnClickListener(this);
      }
}
```

- 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, MainActivity.class).

We will use the Intent object to start the MainActivity greeting activity and pass it information. I will discuss Intents in more depth in later labs. Essentially they are

used to interact with other application components.

We are using Intent.putExtra (<key>, <value>) method to pass the user entered name to the Lab1Activity 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.

<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 finished code after these steps looks like this:

```
public void onClick(View arg0) {
          Intent myIntent = new Intent(this,MainActivity.class);
          myIntent.putExtra("uname",name.getText().toString());

     }
```

- Next, make a call to this.startActivity(**<intent>**);

```
public void onClick(View arg0) {
          Intent myIntent = new Intent(this,Lab1Activity.class);
          myIntent.putExtra("uname",name.getText().toString());

          this.startActivity(myIntent);
     }
```

This command will initiate the switch to the MainActivity greeting Activity.

8. Reconfigure the Hello 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 the application to use the new activity on startup instead of the Main 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 equal from ".MainActivity" to the .GetName activity.

  android:name="itu.edu.hello.GetName"

- 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.

- 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="MainActivity" ></activity>

  This declares to the Android device that the application has an Activity component named MainActivity. If you don't add this tag, Android will not let you launch this Activity.

  The finished **AndroidManifest.xml** file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="itu.edu.hello"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="16" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="itu.edu.hello.GetName"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

```
        <activity android:name="MainActivity" ></activity>

    </application>

</manifest>
```

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 Main Activity greeting activity should appear.

9. Greeting the User

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

After this part the method looks like this:

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Bundle myInput = this.getIntent().getExtras();
    }
```

- Set a reference to the **TextView** object in your **main.xml** layout file.

```
android:id="@+id/textView2"
```

activity_main.xml now looks like:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```

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.

```
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Bundle myInput = this.getIntent().getExtras();

        TextView t=new TextView(this);
    t=(TextView)findViewById(R.id.textView2);
    t.setText("Hello " + (myInput.getString("uname")));
     }
```

Run your application! All should be working now!

10. Exporting Your Application

You can now copy the .apk file created in the /bin directory to your android device. Connect your android device as if it were a USB drive. Drag the file over to the device. Disconnect the device. On the device, select a “file manager” to view the files on the SDK card. Click on the file and it will auto install for you.