DEPARTMENT OF ELECTRONIC ENGINEERING

EE4990 SUMMER TRAINING CAMP FOR OVERSEAS STUDENTS

Forewords

This lab manual is a 2-week self-paced exercise. It is designed as an essential guide to help you quickly get started with Java programming and Android application development. Some of the details are left over in the original text and examples that are authored by Oracle and Google. They are modulated and re-organized in different sections to present a step-by-step learning trail to you. A list of references can be found at the end of each section.

Because Android applications are written using Java, the Android system is more friendly to developers with prior-knowledge of Java. The examples used here are trying to be as simple as possible in such a way that it is suitable for beginners. Normally, you can just follow the instructions straightly to complete the exercises. However, if you want to have a better grasp, and need to integrate and apply what you learnt for your own project development afterwards, then you are advised to go through the two supplementary readings below for Java and XML. Now you can jump to the next section directly. When you have doubts at any point in this tutorial, you can come back to check out these supplementary references again.

1. Java - http://download.oracle.com/javase/tutorial/java/index.html
2. XML - http://www.w3schools.com/xml/default.asp

When you're ready to learn more, read Application Fundamentals for an introduction to all the elements that make Android applications work. Also refer to the API Guide for an overview of the rich Android application framework.

1. Application Fundamentals - http://developer.android.com/guide/components/fundamentals.html
2. API Guide - http://developer.android.com/guide/index.html

Introduction to Java

Getting Started

This trail provides everything you will need to know about getting started with the Java programming language. You will have an overview of Java technology and lessons on installing Java development software and using it to create a simple program.

http://docs.oracle.com/javase/tutorial/getStarted/index.html

(Finish this trail by week 1 day 1)

Object-Oriented Programming Concepts

If you have never used an object-oriented programming language before, you will need to learn a few basic concepts before you can begin writing any code. This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language.

http://docs.oracle.com/javase/tutorial/java/concepts/index.html

(Finish this trail by week 1 day 3)

Language Basics

This trail describes the traditional features of the language, including variables, arrays, data types, operators, and control flow.

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

(Finish this trail by week 1 day 4)

Classes and Objects

With the knowledge you now have of the basics of the Java programming language, you can learn to write your own classes. In this lesson, you will find information about defining your own classes, including declaring member variables, methods, and constructors. You will learn to use your classes to create objects, and how to use the objects you create. This lesson also covers nesting classes within other classes, and enumerations

http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

You can skip **Lambda Expression that is a very new feature introduced in JDK 8.*

(Finish this trail by week 1 day 5)

Interfaces and Inheritance

This trail explains interfaces — what they are, why you would want to write one, and how to write one. This section also describes the way in which you can derive one class from another. That is, how a subclass can inherit fields and methods from a superclass. You will learn that all classes are derived from the Object class, and how to modify the methods that a subclass inherits from superclasses.

http://docs.oracle.com/javase/tutorial/java/IandI/index.html

Numbers and Strings describes how to use Number and String objects The lesson also shows you how to format data for output.

http://docs.oracle.com/javase/tutorial/java/data/index.html

(Finish this trail by week 2 day 1)

Application Development Environment

To develop Android application, you will need:

1. JDK (Java Development Kit)
2. Android SDK
3. Eclipse IDE
4. Android Development Tools (ADT) Plugin for Eclipse

When you are working in our laboratory, the computers are already configured properly and installed the software you need. You may also want to set up the same development environment in your own computer. The link below provides detailed information for you to do the installation.

Android SDK - http://developer.android.com/sdk/index.html

Please note that this tutorial assumes you are building applications for Android 2.2 platform (Froyo, API level 8). Android applications are forward compatible. That means an application built against API level 8 can run on Android 2.2 platform and further releases. If you compile the project in your own computer, you should make sure you have the correct SDK version installed.

Create an AVD

In this tutorial, you will run your application in the Android Emulator. Before you can launch the emulator, you must create an Android Virtual Device (AVD). An AVD defines the system image and device settings used by the emulator. To create an AVD:

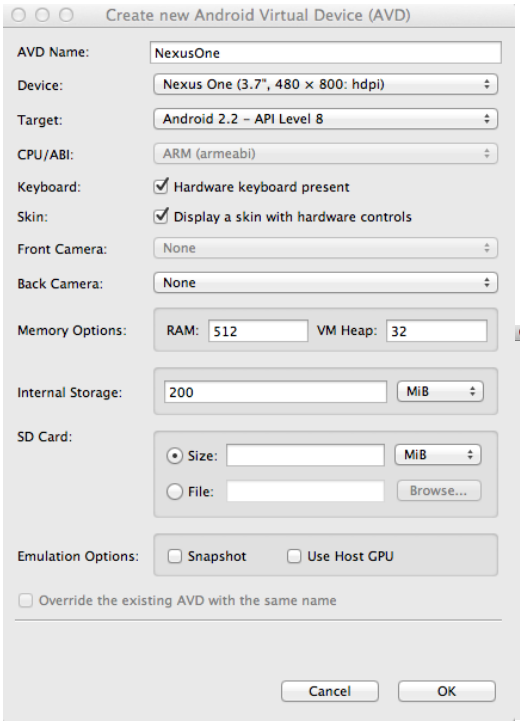
To create an AVD:

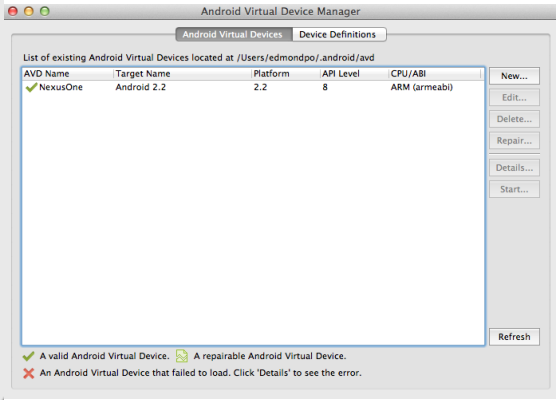
- In Eclipse, choose **Window > Android Virtual Device Manager.**
- Click **New**. The **Create New AVD** dialog appears and create a AVD with following configuration:
    - AVD Name: **NexusOne**
    - Device: **Nexus One (3.7", 480 x 800: hdpi)**
    - Target: Android 2.2 – API Level 8
        - The target is the platform you want to run on the emulator and this Android version 2.2 Froyo that supported by more than 97% of

Android devices.

- You can ignore the rest of the fields for now.

- Click "**OK**" to create this AVD.





Now your environment is ready for development. When you are working in our laboratory, the computers may already configured properly and installed the software you need.

Hello World

As a developer, you know that the first impression of a development framework is how easy it is to write "Hello, World." On Android, it's pretty easy if you're using Eclipse as your IDE, because it has a great plugin that handles your project creation and management to greatly speed-up your development cycles.

Create a New Android Project

In Eclipse, to start a new Android project as follows:

- From Eclipse, select **File > New > Android Application Project**, then a “New Android Application” window will pop up
- Fill in the project details with the following values:
  - Application name : **Hello Android**
  - Project name : **HelloAndroid**
  - Package name : **com.example.helloandroid**
  - Minmum Required SDK: **API 8: Android 2.2 (Froyo)**
  - Target SDK: **API 19: Android 4.4 (KitKat)**
  - Compile With: **API 19: Android 4.4 (KitKat)**
  - Theme: **None**
- Click **Next >** and the following two settings.
- In the Create Activity setting Windows use:
  - Activity name: **MainActivity**
  - Layout name: **activity_main**

Here is a description of each field:

- **Application Name** is the app name that appears to users. For this project, use "Hello Android"
- **Project Name** is the name of your project directory and the name visible in Eclipse.
- **Package Name** is the package namespace for your app (following the same rules as packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. For this reason, it's generally best if you use a name that begins with the reverse domain name of your organization or publisher entity. For this project, you can use something like "com.example.helloandroid." However, you cannot publish your app on Google Play using the "com.example" namespace.

- **Minimum Required SDK** is the lowest version of Android that your app supports, indicated using the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in Supporting Different Platform Versions). Leave this set to the default value for this project.
- **Target SDK** indicates the highest version of Android (also using the API level) with which you have tested with your application. As new versions of Android become available, you should test your app on the new version and update this value to match the latest API level in order to take advantage of new platform features.
- **Compile With** is the platform version against which you will compile your app. By default, this is set to the latest version of Android available in your SDK. (It should be Android 4.1 or greater; if you don't have such a version available, you must install one using the SDK Manager). You can still build your app to support older versions, but setting the build target to the latest version allows you to enable new features and optimize your app for a great user experience on the latest devices.
- **Theme specifies** the Android UI style to apply for your app. You can leave this alone.
- **Create Activity** – This is the name for the class stub that will be generated by the plugin. This will be a subclass of Android's Activity class. An Activity is simply a class that can run and do work. It can create a UI if it chooses, but it doesn't need to. As the checkbox suggests, this is optional, but an Activity is almost always used as the basis for an application.

Your Android project is now ready. It should be visible in the Package Explorer on the left. Open the MainActivity.java file, located inside

HelloAndroid > src > com.example.helloandroid.

It should look like this:

```
package com.example.helloworld;

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

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
```

```
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

Notice that the class is based on the Activity class. An Activity is a single application entity that is used to perform actions. An application may have many separate activities, but the user interacts with them one at a time. The onCreate() method will be called by the Android system when your Activity starts — it is where you should perform all initialization and UI setup. An activity is not required to have a user interface, but usually will. Now let's modify some code!

Construct the UI Programmatically

Take a look at the revised code below and then make the same changes to your MainActivity class. The bold items are lines that have been added.

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

Tip: There are some useful Eclipse shortcuts that you may use very often:

- **Ctrl-Shift-O** – Automatically identify missing packages based on your code and add them for you.
- **Ctrl-Shift-F** – Automatically format your code nicely (i.e. spacing and indentation).
- **Ctrl-Shift-S** – Save!

An Android user interface is composed of hierarchies of objects called Views. A View is a drawable object used as an element in your UI layout, such as a button, image, or (in this case) a text label. Each of these objects is a subclass of the View class and the subclass that handles text is TextView.

In this change, you create a TextView with the class constructor, which accepts an Android Context instance as its parameter. A Context is a handle to the system; it provides services like resolving resources, obtaining access to databases and preferences, and so on. The Activity class inherits from Context, and because your HelloAndroid class is a subclass of Activity, it is also a Context. So, you can pass this as your Context reference to the TextView.

Next, you define the text content with setText().

Finally, you pass the TextView to setContentView() in order to display it as the content for the Activity UI. If your Activity doesn't call this method, then no UI is present and the system will display a blank screen.

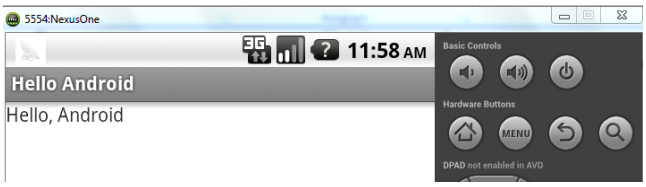
There it is — "Hello, World" in Android! The next step, of course, is to see it running.

Run the Application

The Eclipse plugin makes it easy to run your applications:

1. Select **Run > Run**.
2. Select "**Android Application**".

The Eclipse plugin automatically creates a new run configuration for your project and then launches the Android Emulator. Depending on your environment, the Android emulator might take several minutes to boot fully, so please be patient. When the emulator is booted, the Eclipse plugin installs your application and launches the default Activity. You should now see something like this:



The "Hello Android" you see in the grey bar is actually the application title. The Eclipse plugin creates this automatically (the string is defined in the res/values/strings.xml file and referenced by your AndroidManifest.xml file). The text below the title is the actual text that you have created in the TextView object.

That concludes the basic "Hello World" tutorial, but you should continue reading for some more valuable information about developing Android applications.

Upgrade the UI to an XML Layout

The "Hello, World" example you just completed uses what is called a "programmatic" UI layout. This means that you constructed and built your application's UI directly in source code. If you've done much UI programming, you're probably familiar with how brittle that approach can sometimes be: small changes in layout can result in big source-code headaches. It's also easy to forget to properly connect Views together, which can result in errors in your layout and wasted time debugging your code.

That's why Android provides an alternate UI construction model: XML-based layout files. The easiest way to explain this concept is to show an example. Here's an XML layout file that is identical in behavior to the programmatically-constructed example. Open the activity_main.xml file located inside,

HelloAndroid > res > layout > activity_main.xml

```
<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"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity horizontal margin"
    android:paddingRight="@dimen/activity horizontal margin"
    android:paddingTop="@dimen/activity vertical margin"
    tools:context=".MainActivity" >


    <TextView
        android:layout width="wrap content"
        android:layout height="wrap content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

The general structure of an Android XML layout file is simple: it's a tree of XML elements, wherein each node is the name of a View class (this example, however, is just one View element). You can use the name of any class that extends View as an element in your XML layouts, including custom View classes you define in your own code. This structure makes it easy to quickly build up UIs, using a more simple structure and syntax than you would use in a programmatic layout. This model is inspired by the web development model, wherein you can separate the presentation of your application (its UI) from the application logic used to fetch and fill in data.

In the above XML example, there's just one View element: the TextView, which has five XML attributes. Here's a summary of what they mean:

- **xmlns:android** - This is an XML namespace declaration that tells the Android tools that you are going to refer to common attributes defined in the Android namespace. The outermost tag in every Android layout file must have this attribute.
- **android:id** - This attribute assigns a unique identifier to the TextView element. You can use the assigned ID to reference this View from your source code or from other XML resource declarations.
- **android:layout_width** - This attribute defines how much of the available width on the screen this View should consume. In this case, it's the only View so you want it to take up the entire screen, which is what a value of "fill_parent" means.
- **android:layout_height** - This is just like android:layout_width, except that it refers to available screen height.
- **android:text** - This sets the text that the TextView should display. In this example, you use a string resource instead of a hard-coded string value. The hello string is defined in the res/values/strings.xml file. This is the recommended practice for inserting strings to your application, because it makes the localization of your application to other languages graceful, without need to hard-code changes to the layout file.

These XML layout files belong in the res/layout/ directory of your project. The "res" is short for "resources" and the directory contains all the non-code assets that your application requires. In addition to layout files, resources also include assets such as images, sounds, and localized strings.

The Eclipse plugin automatically creates one of these layout files for you: main.xml. In the "Hello World" application you just completed, this file was ignored and you created a layout programmatically. This was meant to teach you more about the Android framework, but **you should almost always define your layout in an XML file instead of in your code**. The following procedures will instruct you how to change your existing application to use an XML layout.

1. In the Eclipse Package Explorer, expand the /res/layout/ folder and open activity_main.xml (once opened, you might need to click the "activity_main.xml" tab at the bottom of the window to see the XML source). Replace the contents with the XML layout just defined above.
2. Inside the res/values/ folder, open strings.xml. This is where you should save all default text strings for your user interface. If you're using Eclipse, then ADT will have started

you with two strings, hello and app_name. Revise hello to something else. Perhaps "Hello, Android! I am a string resource!" The entire file should now look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Hello Android</string>
    <string name="action_settings">Settings</string>
    <string name="hello world"> Hello, Android! I am a string resource!</string>

</resources>
```

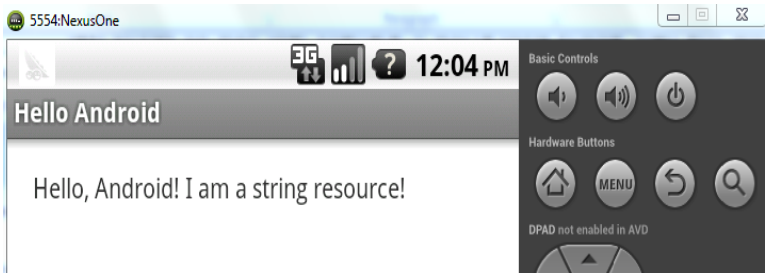
3. Now open and modify your MainActivity class to use the XML layout. Edit the file to look like this:

```
package com.example.helloandroid;

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

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

When you make this change, type it by hand to try the code-completion feature. As you begin typing "R.layout.activity_main" the plugin will offer you suggestions. You'll find that it helps in a lot of situations.

Instead of passing setContentView() a View object, you give it a reference to the layout resource. The resource is identified as R.layout.activity_main, which is actually a compiled object representation of the layout defined in /res/layout/activity_main.xml. The Eclipse plugin automatically creates this reference for you inside the project's R.java class.

Now re-run your application — because you've created a launch configuration, all you need to do is click the green arrow icon to run, or select **Run > Run History > Android Activity**. Other than the change to the TextView string, the application looks the same. After all, the point was to show that the two different layout approaches produce identical results.



The R Class

In Eclipse, open the file named R.java (in the gen/ [Generated Java Files] folder). It should look something like this:

```
package com.example.helloandroid;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int textview=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

A project's R.java file is an index into all the resources defined in the file. You use this class in your source code as a sort of short-hand way to refer to resources you've included in your project. This is particularly powerful with the code-completion features of IDEs like Eclipse because it lets you quickly and interactively locate the specific reference you're looking for.

It's possible yours looks slightly different than this (perhaps the hexadecimal values are different). For now, notice the inner class named "layout", and its member field "main". The Eclipse plugin noticed the XML layout file named main.xml and automatically generated a class for it here. As you add other resources to your project (such as strings in the res/values/string.xml file or drawables inside the res/drawable/direcory) you'll see R.java change to keep up. **You should never edit this file by hand.**

CHECKPOINT 1 _____ *(Signed by Demonstrator)*

- Demonstrate your upgraded “Hello World” program as shown in p.12.
- Change the app’s name to “Hello EE4990” and show it again.

Hello Layout

Your layout is the architecture for the user interface in an Activity. It defines the layout structure and holds all the elements that appear to the user. You can declare your layout in two ways:

1. **Declare UI elements in XML.** Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
2. **Instantiate layout elements at runtime.** Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

The Android framework gives you the flexibility to use either or both of these methods for declaring and managing your application's UI. For example, you could declare your application's default layouts in XML, including the screen elements that will appear in them and their properties. You could then add code in your application that would modify the state of the screen objects, including those declared in XML, at run time.

The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior. Your UI descriptions are external to your application code, which means that **you can modify or adapt it without having to modify your source code and recompile**. For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems. As such, the following exercises focus on teaching you how to declare common layouts in XML.

Linear Layout

LinearLayout is a ViewGroup that displays child View elements in a linear direction, either vertically or horizontally. You should be careful about over-using the LinearLayout. If you begin nesting multiple LinearLayouts, you may want to consider using a RelativeLayout instead (see next section).

1. Start a new project named HelloLayout.
2. Follow the hello-world-style to fill in these values:

   - Application name : **Hello Layout**

- Project name :**HelloLayout**
- Package name : **com.example.hellolayout**
- Minmum Required SDK: **API 8: Android 2.2 (Froyo)**
- Target SDK: **API 19: Android 4.4 (KitKat)**
- Compile With: **API 19: Android 4.4 (KitKat)**
- Theme: **None**

3. Create an XML file named linear_view.xml and save it inside the res/layout/ folder. Insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent">

  <LinearLayout
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout height="fill parent"
      android:layout weight="1">
      <TextView
          android:text="red"
          android:gravity="center_horizontal"
          android:background="#aa0000"
          android:layout width="wrap content"
          android:layout height="fill parent"
          android:layout_weight="1"/>
      <TextView
          android:text="green"
          android:gravity="center horizontal"
          android:background="#00aa00"
          android:layout width="wrap content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="blue"
          android:gravity="center horizontal"
          android:background="#0000aa"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
      <TextView
          android:text="yellow"
          android:gravity="center_horizontal"
          android:background="#aaaa00"
          android:layout width="wrap content"
          android:layout height="fill parent"
          android:layout weight="1"/>
  </LinearLayout>

  <LinearLayout
    android:orientation="vertical"
    android:layout width="fill parent"
    android:layout height="fill parent"
    android:layout_weight="1">
    <TextView
        android:text="row one"
        android:textSize="15pt"
        android:layout width="fill parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="row two"
        android:textSize="15pt"
        android:layout width="fill parent"
        android:layout_height="wrap_content"
```

```
        android:layout_weight="1"/>
    <TextView
        android:text="row three"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView
        android:text="row four"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
  </LinearLayout>

</LinearLayout>
```

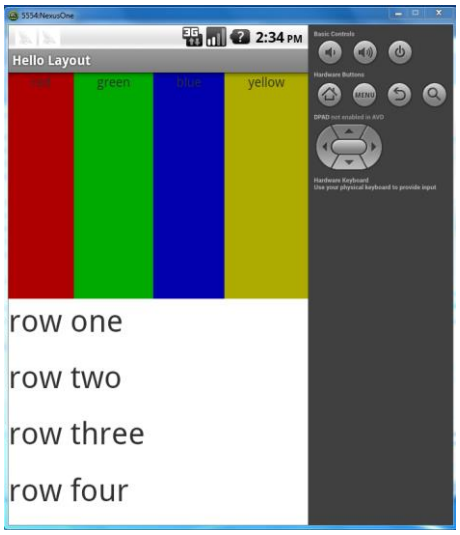
Carefully inspect this XML. There is a root LinearLayout that defines its orientation to be vertical—all child Views (of which it has two) will be stacked vertically. The first child is another LinearLayout that uses a horizontal orientation and the second child is a LinearLayout that uses a vertical orientation. Each of these nested LinearLayouts contains several TextView elements, which are oriented with each other in the manner defined by their parent LinearLayout.

4. The default res/layout/activity_main.xml file is no use now. Instead, open MainActivity.java and update it to load the layout res/layout/linear_view.xml file in the onCreate() method:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.linear_view);
}
```

The setContentView(int) method loads the layout file for the Activity, specified by the resource ID — R.layout.linear_view refers to the res/layout/linear_view.xml layout file.

5. Run the application. You should see the following:



Notice how the XML attributes define each View's behavior. Try experimenting with different values for android:layout_weight to see how the screen real estate is distributed based on the weight of each element. The legal values for this attribute are documented here:

- http://developer.android.com/guide/topics/resources/layout-resource.html

Relative Layout

RelativeLayout is a ViewGroup that displays child View elements in relative positions. The position of a View can be specified as relative to sibling elements (such as to the left-of or below a given element) or in positions relative to the RelativeLayout area (such as aligned to the bottom, left of center).

A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested ViewGroups. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.

1. Continue the HelloLayout project. Create an XML file named relative_view.xml and save it inside the res/layout/ folder. Insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

Notice each of the android:layout_* attributes, such as layout_below, layout_alignParentRight, and layout_toLeftOf. When using a RelativeLayout, you can use these attributes to describe how you want to position each View. Each one of these attributes defines a different kind of relative position. Some attributes use the resource ID of a sibling View to define its own relative position. For example, the last Button is defined to lie to the left-of and aligned-with-the-top-of the View identified by the ID ok (which is the previous Button). All of the available layout attributes for RelativeLayout are defined in RelativeLayout.LayoutParams.

- http://developer.android.com/reference/android/widget/RelativeLayout.LayoutParams.html

2. Make sure you load this layout in the onCreate() method. Change the resource ID to R.layout.relative_view that refers to the res/layout/relative_view.xml layout file.
3. Run the applications. You should see the following:

Table Layout

TableLayout is a ViewGroup that displays child View elements in rows and columns.

1. Continue the HelloLayout project. Create an XML file named table_view.xml and save it inside the res/layout/ folder. Insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
```

```
    <TableRow>
        <TextView
            android:layout column="1"
            android:text="Open..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout column="1"
            android:text="Save..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout column="1"
            android:text="Save As..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <View
        android:layout height="2dip"
        android:background="#FF909090" />

    <TableRow>
        <TextView
            android:text="X"
            android:padding="3dip" />
        <TextView
            android:text="Import..."
            android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:text="X"
            android:padding="3dip" />
        <TextView
            android:text="Export..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-E"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <View
        android:layout_height="2dip"
        android:background="#FF909090" />

    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Quit"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

Notice how this resembles the structure of an HTML table. The TableLayout element is like the HTML <table> element; TableRow is like a <tr> element; but for the cells, you can use

any kind of View element. In this example, a TextView is used for each cell. In between some of the rows, there is also a basic View, which is used to draw a horizontal line

2. Make sure you load this layout in the onCreate() method. Change the resource ID to R.layout.table_view that refers to the res/layout/table_view.xml layout file.
3. Run the applications. You should see the following:

References

- Linear Layout - http://developer.android.com/guide/topics/ui/layout/linear.html
- Relative Layout - http://developer.android.com/guide/topics/ui/layout/relative.html

To see more common layout objects, find the document below:

- Common Layout Objects - http://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts

Hello Widget

A widget is a View object that serves as an interface for interaction with the user. Android provides a set of fully implemented widgets, like buttons, checkboxes, and text-entry fields, so you can quickly build your UI. Some widgets provided by Android are more complex, like a date picker, a clock, and zoom controls. But you're not limited to the kinds of widgets provided by the Android platform. If you'd like to do something more customized and create your own actionable elements, you can, by defining your own View object or by extending and combining existing widgets.

Form Stuff

This part introduces a variety of widgets that are useful when creating forms, such as image buttons, text fields, checkboxes and radio buttons.

1. Start a new project named **HelloFormStuff**.
2. Your res/layout/activity_main.xml file should already have a basic LinearLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

</LinearLayout>
```

For each widget you are going to add, just put the respective View inside this Linear Layout.

Each section below also assumes that your HelloFormStuff Activity has the following default implementation of the onCreate() method:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

Custom Button

In this section, you will create a button with a custom image instead of text, using the Button widget and an XML file that defines three different images to use for the different button states. When the button is pressed, a short message will be displayed.

1. Copy the three images into the res/drawable-mdpi/ directory of your project. These will be used for the different button states.
   - android_focused.png,
   - android_normal.png
   - android_pressed.png
2. Create a new folder of res/drawable/ with a new XML file named **android_button.xml** in this folder. Insert the following XML code in this file (res/drawable/android_button.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/android pressed"
          android:state pressed="true" />
    <item android:drawable="@drawable/android focused"
          android:state_focused="true" />
    <item android:drawable="@drawable/android_normal" />
</selector>
```

This defines a single drawable resource, which will change its image based on the current state of the button. The first <item> defines android_pressed.png as the image when the button is pressed (it's been activated); the second <item> defines android_focused.png as the image when the button is focused (when the button is highlighted using the trackball or directional pad); and the third <item> defines android_normal.png as the image for the normal state (when neither pressed nor focused). This XML file now represents a single drawable resource and when referenced by a Button for its background, the image displayed will change based on these three states.

3. Open the res/layout/activity_main.xml file and add the Button element:

```
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:background="@drawable/android_button" />
```

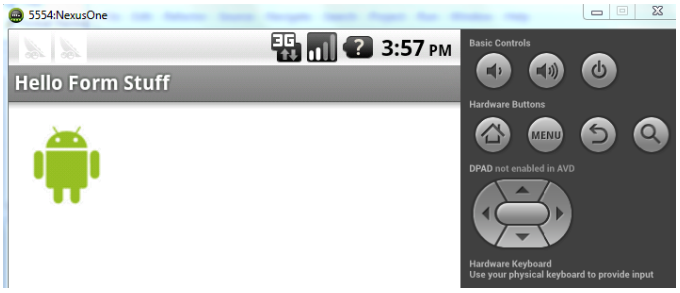
The android:background attribute specifies the drawable resource to use for the button background (which, when saved at res/drawable/android_button.xml, is referenced as @drawable/android_button). This replaces the normal background image used for buttons throughout the system. In order for the drawable to change its image based on the button state, the image must be applied to the background.

4. To make the button do something when pressed, add the following code at the end of the onCreate() method:

```
final Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        Toast.makeText(MainActivity.this, "Beep Bop", Toast.LENGTH_SHORT).show();
    }
});
```

This captures the Button from the layout, then adds an View.OnClickListener. The View.OnClickListener must implement the onClick(View) callback method, which defines the action to be made when the button is clicked. In this example, a Toast message will be displayed.

5. Now run the application and press the image button to experience the effect of changing the image of the button.



Edit Text

In this section, you will create a text field for user input, using the EditText widget. Once text has been entered into the field, the "Enter" key will display the text in a toast message.

1. Open the res/layout/activity_main.xml file and add the EditText element (inside the LinearLayout):

```
<EditText
        android:id="@+id/edittext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
```

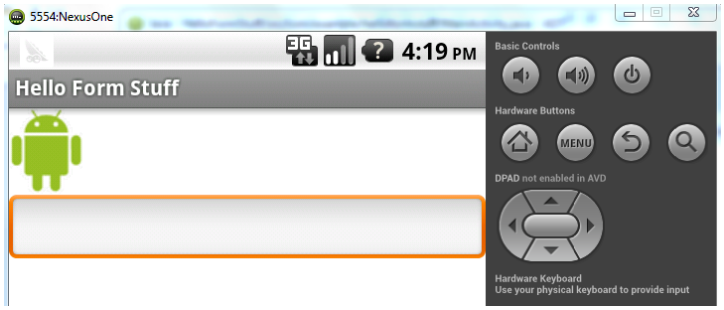
2. To do something with the text that the user types, add the following code to the end of the onCreate() method:

```
final EditText edittext = (EditText) findViewById(R.id.edittext);
edittext.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        // If the event is a key-down event on the "enter" button
        if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
            (keyCode == KeyEvent.KEYCODE_ENTER)) {
          // Perform action on key press
```

```
            Toast.makeText(MainActivity.this, edittext.getText(),
Toast.LENGTH_SHORT).show();
            return true;
        }
        return false;
    }
});
```

This captures the EditText element from the layout and adds an View.OnKeyListener. The View.OnKeyListener must implement the onKey(View, int, KeyEvent) method, which defines the action to be made when a key is pressed while the widget has focus. In this case, the method is defined to listen for the Enter key (when pressed down), then pop up a Toast message with the text that has been entered. The onKey(View, int, KeyEvent) method should always return true if the event has been handled, so that the event doesn't bubble-up (which would result in a carriage return in the text field).

3. Run the application.



Check Box

In this section, you will create a checkbox for selecting items, using the CheckBox widget. When the checkbox is pressed, a toast message will indicate the current state of the checkbox.

1. Open the res/layout/layout_main.xml file and add the CheckBox element (inside the LinearLayout):

```
<CheckBox android:id="@+id/checkbox"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="check it out" />
```

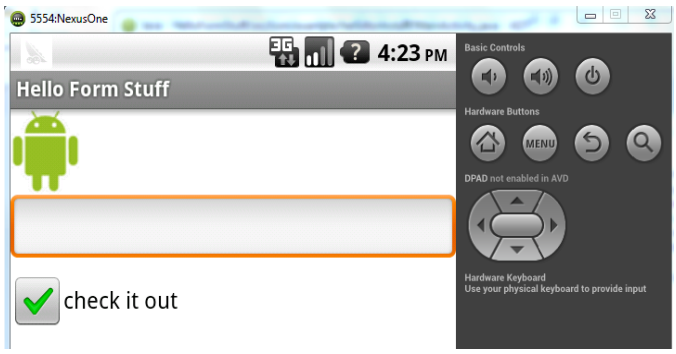
2. To do something when the state is changed, add the following code to the end of the onCreate() method:

```
final CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox);
checkbox.setOnClickListener(new OnClickListener() {
```

```
    public void onClick(View v) {
        // Perform action on clicks, depending on whether it's now checked
        if (((CheckBox) v).isChecked()) {
            Toast.makeText(MainActivity.this, "Selected", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(MainActivity.this, "Not selected",
Toast.LENGTH_SHORT).show();
        }
    }
});
```

This captures the CheckBox element from the layout, then adds an View.OnClickListener. The View.OnClickListener must implement the onClick(View) callback method, which defines the action to be made when the checkbox is clicked. When clicked, isChecked() is called to check the new state of the check box. If it has been checked, then a Toast displays the message "Selected", otherwise it displays "Not selected". Note that the View object that is passed in the onClick(View) callback must be cast to a CheckBox because the isChecked() method is not defined by the parent View class. The CheckBox handles its own state changes, so you only need to query the current state.

3. Run it.



Radio Button

In this section, you will create two mutually-exclusive radio buttons (enabling one disables the other), using the RadioGroup and RadioButton widgets. When either radio button is pressed, a toast message will be displayed.

1. Open the res/layout/activity_main.xml file and add two RadioButtons, nested in a RadioGroup (inside the LinearLayout):

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_red"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="Red" />
        <RadioButton android:id="@+id/radio_blue"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Blue" />
    </RadioGroup>
```

It's important that the RadioButtons are grouped together by the RadioGroup element so that no more than one can be selected at a time. This logic is automatically handled by the Android system. When one RadioButton within a group is selected, all others are automatically deselected.

2. To do something when each RadioButton is selected, you need a View.OnClickListener. In this case, you want the listener to be re-usable by the two buttons, so add the following code to create a new member in the MainActivity class:

```
OnClickListener radio_listener = new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        RadioButton rb = (RadioButton) v;
        Toast.makeText(MainActivity.this, rb.getText(), Toast.LENGTH_SHORT).show();
    }
};
```

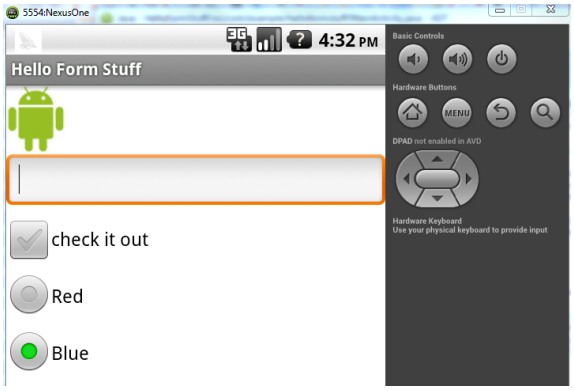
First, the View that is passed to the onClick(View) method is cast into a RadioButton. Then a Toast message displays the selected radio button's text.

3. Now, at the bottom of the onCreate() method, add the following:

```
final RadioButton radio_red = (RadioButton) findViewById(R.id.radio_red);
final RadioButton radio_blue = (RadioButton) findViewById(R.id.radio_blue);
radio_red.setOnClickListener(radio_listener);
radio_blue.setOnClickListener(radio_listener);
```

This captures each of the RadioButtons from the layout and adds the newly-created View.OnClickListener to each of them.

4. Run the application.



Toggle Button

In this section, you'll create a button used specifically for toggling between two states, using the ToggleButton widget. This widget is an excellent alternative to radio buttons if you have two simple states that are mutually exclusive ("on" and "off", for example).

1. Open the res/layout/activity_main.xml file and add the ToggleButton element (inside the LinearLayout):

```
<ToggleButton android:id="@+id/togglebutton"
        android:layout width="wrap content"
        android:layout height="wrap content"
        android:textOn="Vibrate on"
        android:textOff="Vibrate off"/>
```

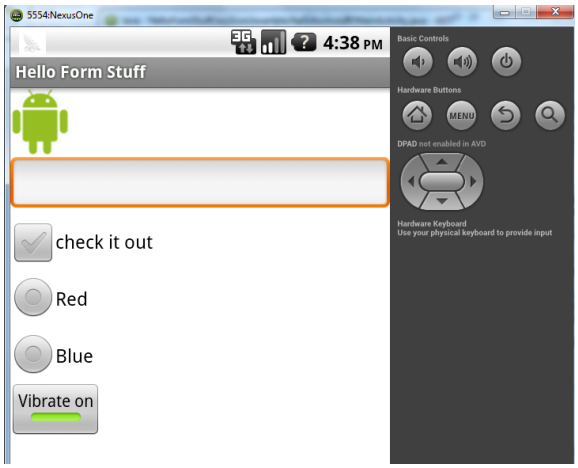
The attributes android:textOn and android:textOff specify the text for the button when the button has been toggled on or off. The default values are "ON" and "OFF".

2. To do something when the state is changed, add the following code to the end of the onCreate() method:

```
final ToggleButton togglebutton = (ToggleButton) findViewById(R.id.togglebutton);
togglebutton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        if (togglebutton.isChecked()) {
            Toast.makeText(MainActivity.this, "Checked", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(MainActivity.this, "Not checked",
Toast.LENGTH SHORT).show();
        }
    }
});
```

This captures the ToggleButton element from the layout, then adds an View.OnClickListener. The View.OnClickListener must implement the onClick(View) callback method, which defines the action to perform when the button is clicked. In this example, the callback method checks the new state of the button, then shows a Toast message that indicates the current state. Notice that the ToggleButton handles its own state change between checked and unchecked, so you just ask which it is.

3. Run the application.



Rating Bar

In this section, you'll create a RatingBar widget that allows the user to provide a rating.

1. Open the res/layout/activity_main.xml file and add the RatingBar element (inside the LinearLayout):

```
    <RatingBar android:id="@+id/ratingbar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:numStars="5"
        android:stepSize="1.0"/>
```

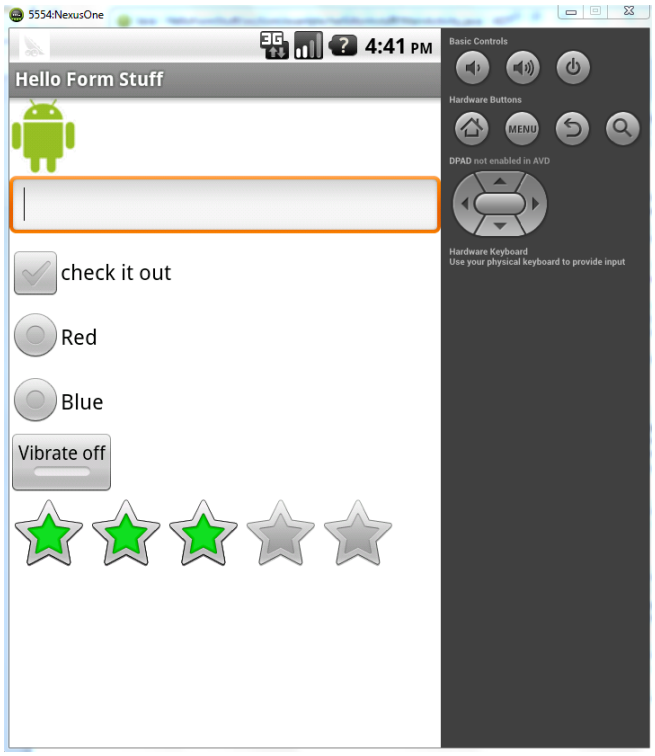
The android:numStars attribute defines how many stars to display for the rating bar. The android:stepSize attribute defines the granularity for each star (for example, a value of 0.5 would allow half-star ratings).

2. To do something when a new rating has been set, add the following code to the end of the onCreate() method:

```
final RatingBar ratingbar = (RatingBar) findViewById(R.id.ratingbar);
ratingbar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {
        Toast.makeText(MainActivity.this, "New Rating: " + rating,
Toast.LENGTH_SHORT).show();
    }
});
```

This captures the RatingBar widget from the layout with findViewById(int) and then sets an RatingBar.OnRatingBarChangeListener. The onRatingChanged() callback method then defines the action to perform when the user sets a rating. In this case, a simple Toast message displays the new rating.

3. Run the application. If you've added all the form widgets above, your application should look like this:



Date Picker

To provide a widget for selecting a date, use the DatePicker widget, which allows the user to select the month, day, and year, in a familiar interface.

In this section, you'll create a DatePickerDialog, which presents the date picker in a floating dialog box at the press of a button. When the date is set by the user, a TextView will update with the new date.

1. Start a new project named HelloDatePicker.
2. Open the res/layout/activity_main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout width="wrap content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView android:id="@+id/dateDisplay"
        android:layout width="wrap content"
        android:layout height="wrap content"
        android:text=""/>
    <Button android:id="@+id/pickDate"
        android:layout_width="wrap_content"
        android:layout height="wrap content"
        android:text="Change the date"/>
</LinearLayout>
```

This creates a basic LinearLayout with a TextView that will display the date and a Button that will open the DatePickerDialog.

3. Open MainActivity.java and add the following members to the class:

```
    private TextView mDateDisplay;
    private Button mPickDate;
    private int mYear;
    private int mMonth;
    private int mDay;

    static final int DATE_DIALOG_ID = 0;
```

The first group of members define variables for the layout Views and the date items. The DATE_DIALOG_ID is a static integer that uniquely identifies the Dialog that will display the date picker.

4. Now add the following code for the onCreate() method:

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

        // capture our View elements
        mDateDisplay = (TextView) findViewById(R.id.dateDisplay);
        mPickDate = (Button) findViewById(R.id.pickDate);
```

```
        // add a click listener to the button
        mPickDate.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                showDialog(DATE DIALOG ID);
            }
        });

        // get the current date
        final Calendar c = Calendar.getInstance();
        mYear = c.get(Calendar.YEAR);
        mMonth = c.get(Calendar.MONTH);
        mDay = c.get(Calendar.DAY_OF_MONTH);

        // display the current date (this method is below)
        updateDisplay();
    }
```

First, the content is set to the activity_main.xml layout. Then the TextView and Button elements are captured from the layout with findViewById(int). A new View.OnClickListener is created for the Button, so that when it is clicked, it will call showDialog(int), passing the unique integer ID for the date picker dialog. Using showDialog(int) allows the Activity to manage the life-cycle of the dialog and will call the onCreateDialog(int) callback method to request the Dialog that should be displayed (which you'll define later). After the on-click listener is set, a new Calendar is created and the current year, month and day are acquired. Finally, the private updateDisplay() method is called in order to fill the TextView with the current date.

5. Add the updateDisplay() method:

```
    // updates the date in the TextView
    private void updateDisplay() {
        mDateDisplay.setText(
            new StringBuilder()
                    // Month is 0 based so add 1
                    .append(mMonth + 1).append("-")
                    .append(mDay).append("-")
                    .append(mYear).append(" "));
    }
```

This method uses the member date values declared for the class to write the date to the layout's TextView, mDateDisplay, which was also declared and initialized above.

6. Initialize a new DatePickerDialog.OnDateSetListener as a member of the HelloDatePicker class:

```
    // the callback received when the user "sets" the date in the dialog
    private DatePickerDialog.OnDateSetListener mDateSetListener =
            new DatePickerDialog.OnDateSetListener() {

                public void onDateSet(DatePicker view, int year,
                                      int monthOfYear, int dayOfMonth) {
                    mYear = year;
                    mMonth = monthOfYear;
                    mDay = dayOfMonth;
                    updateDisplay();
```

```
            }
        };
```

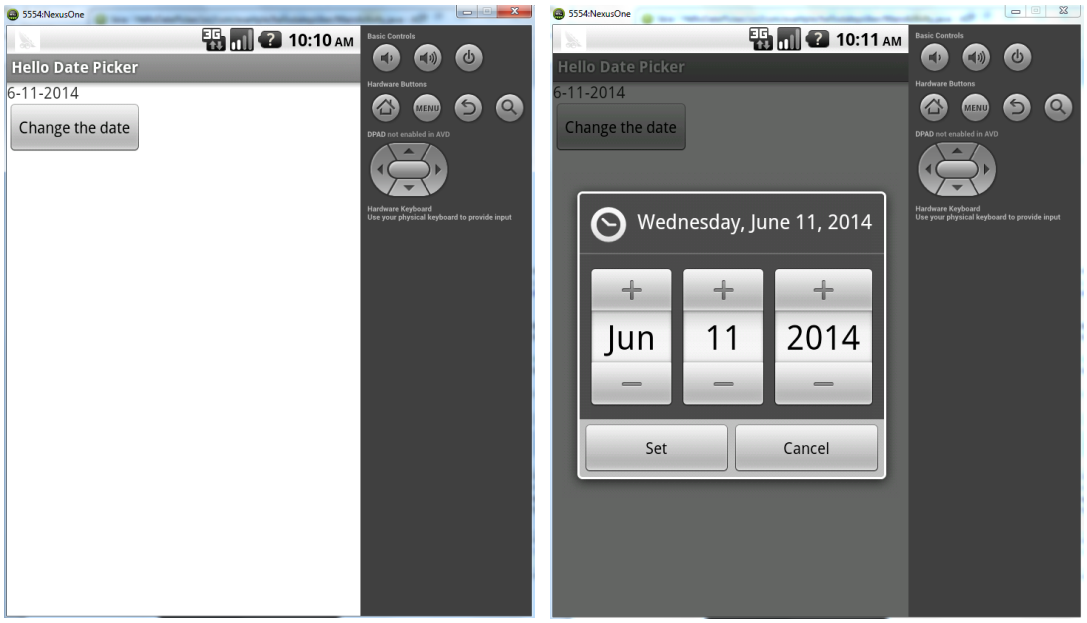
The DatePickerDialog.OnDateSetListener listens for when the user has set the date (by clicking the "Set" button). At that time, the onDateSet() callback method is called, which is defined to update the mYear, mMonth, and mDay member fields with the new date then call the private updateDisplay() method to update the TextView.

7. Now add the onCreateDialog(int) callback method to the MainActivity class:

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
    case DATE_DIALOG_ID:
        return new DatePickerDialog(this,
                    mDateSetListener,
                    mYear, mMonth, mDay);
    }
    return null;
}
```

This is an Activity callback method that is passed the integer ID given to showDialog(int) (which is called by the button's View.OnClickListener). When the ID matches the switch case defined here, a DatePickerDialog is instantiated with the DatePickerDialog.OnDateSetListener created in the previous step, along with the date variables to initialize the widget date.

8. Run the application. When you press the "Change the date" button, you should see the following:



References

- Input Controls - http://developer.android.com/guide/topics/ui/controls.html
- Pickers - http://developer.android.com/guide/topics/ui/controls/pickers.html

Hello AdapterView

The AdapterView is a ViewGroup subclass whose child Views are determined by an Adapter that binds to data of some type. AdapterView is useful whenever you need to display stored data (as opposed to resource strings or drawables) in your layout. ListView, GridView and Spinner are examples of AdapterView subclasses that you can use to bind to a specific type of data and display it in a certain way. AdapterView objects have two main responsibilities:

1. Filling the layout with data
2. Handling user selections

List View

ListView is a ViewGroup that creates a list of scrollable items. The list items are automatically inserted to the list using a ListAdapter.

In this section, you'll create a scrollable list of country names that are read from a string array. When a list item is selected, a toast message will display the position of the item in the list.

1. Start a new project named HelloListView.
2. Create an XML file named list_item.xml and save it inside the res/layout/ folder. Insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

This file defines the layout for each item that will be placed in the ListView.

3. Open the MainActivity.java and make the class extend ListActivity (instead of Activity):

```
public class MainActivity extends ListActivity {
```

4. Insert the following code for the onCreate() method:

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

```
  setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item, COUNTRIES));

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);

  lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
      // When clicked, show a toast with the TextView text
      Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
          Toast.LENGTH_SHORT).show();
    }
  });
}
```

Notice that this does not load a layout file for the Activity (which you usually do with setContentView(int)). Instead, setListAdapter(ListAdapter) automatically adds a ListView to fill the entire screen of the ListActivity. This method takes an ArrayAdapter, which manages the array of list items that will be placed into the ListView. The ArrayAdapter constructor takes the application Context, the layout description for each list item (created in the previous step), and a List of objects to insert in the ListView (defined next).

The setTextFilterEnabled(boolean) method turns on text filtering for the ListView, so that when the user begins typing, the list will be filtered.

The setOnItemClickListener(OnItemClickListener) method defines the on-click listener for each item. When an item in the ListView is clicked, the onItemClick() method is called and a Toast message is displayed, using the text from the clicked item.

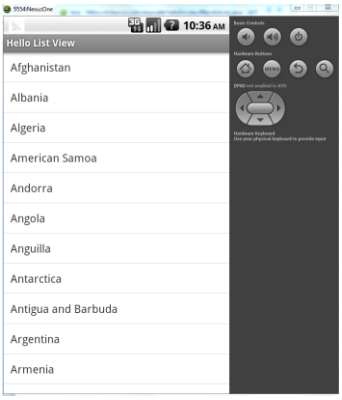
Tip: You can use list item designs provided by the platform instead of defining your own layout file for the ListAdapter. For example, try using android.R.layout.simple_list_item_1 instead of R.layout.list_item.

5. After the onCreate() method, add the string array:

```
static final String[] COUNTRIES = new String[] {
    "Afghanistan", "Albania", "Algeria", "American Samoa", "Andorra",
    "Angola", "Anguilla", "Antarctica", "Antigua and Barbuda", "Argentina",
    "Armenia", "Aruba", "Australia", "Austria", "Azerbaijan",
    "Bahrain", "Bangladesh", "Barbados", "Belarus", "Belgium",
    "Belize", "Benin", "Bermuda", "Bhutan", "Bolivia",
    "Bosnia and Herzegovina", "Botswana", "Bouvet Island", "Brazil",
    "British Virgin Islands", "Brunei", "Bulgaria", "Burkina Faso", "Burundi",
    "Cote d'Ivoire", "Cambodia", "Cameroon", "Canada", "Cape Verde",
    "Cayman Islands", "Central African Republic", "Chad", "Chile", "China",
    "Christmas Island", "Cocos (Keeling) Islands", "Colombia", "Comoros", "Congo",
    "Cook Islands", "Costa Rica", "Croatia", "Cuba", "Cyprus", "Czech Republic",
    "Democratic Republic of the Congo", "Denmark", "Djibouti", "Dominica",
    "East Timor", "Ecuador", "Egypt", "El Salvador", "Equatorial Guinea", "Eritrea",
    "Estonia", "Ethiopia"
  };
```

This is the array of strings that will be placed into the ListView.

6. Run the application. You can scroll the list, or type to filter it, then click an item to see a message. You should see something like this:



Note that using a hard-coded string array is not the best design practice. One is used in this tutorial for simplicity, in order to demonstrate the ListView widget. The better practice is to reference a string array defined by an external resource, such as with a <string-array> resource in your project res/values/strings.xml file. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries array">
        <item>Bahrain</item>
        <item>Bangladesh</item>
        <item>Barbados</item>
        <item>Belarus</item>
        <item>Belgium</item>
        <item>Belize</item>
        <item>Benin</item>
    </string-array>
</resources>
```

To use these resource strings for the ArrayAdapter, replace the original setListAdapter(ListAdapter) line with the following:

```
String[] countries = getResources().getStringArray(R.array.countries_array);
setListAdapter(new ArrayAdapter<String>(this, R.layout.list_item, countries));
```

Grid View

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a ListAdapter.

In this section, you'll create a grid of image thumbnails. When an item is selected, a toast message will display the position of the image.

1. Start a new project named HelloGridView.
2. Save the sample image files (sample_*.jpg) into the project's res/drawable-mdpi/ directory.
3. Open the res/layout/activity_main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

This GridView will fill the entire screen. The attributes are rather self explanatory. For more information about valid attributes, see the GridView reference.

4. Open MainActivity.java and insert the following code for the onCreate() method:

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

    GridView gridview = (GridView) findViewById(R.id.gridview);
    gridview.setAdapter(new ImageAdapter(this));

    gridview.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View v, int position, long id)
{
            Toast.makeText(MainActivity.this, "" + position,
Toast.LENGTH_SHORT).show();
        }
    });
}
```

After the activity_main.xml layout is set for the content view, the GridView is captured from the layout with findViewById(int). The setAdapter() method then sets a custom adapter (ImageAdapter) as the source for all items to be displayed in the grid. The ImageAdapter is created in the next step.

To do something when an item in the grid is clicked, the setOnItemClickListener() method is passed a new AdapterView.OnItemClickListener. This anonymous instance defines the onItemClick() callback method to show a Toast that displays the index position (zero-based) of the selected item (in a real world scenario, the position could be used to get the full sized image for some other task).

5. Create a new class called ImageAdapter that extends BaseAdapter:

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {  // if it's not recycled, initialize some attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = {
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7,
            R.drawable.sample_0, R.drawable.sample_1,
            R.drawable.sample_2, R.drawable.sample_3,
            R.drawable.sample_4, R.drawable.sample_5,
            R.drawable.sample_6, R.drawable.sample_7
    };
}
```

First, this implements some required methods inherited from BaseAdapter. The constructor and getCount() are self-explanatory. Normally, getItem(int) should return the actual object at the specified position in the adapter, but it's ignored for this example. Likewise, getItemId(int) should return the row id of the item, but it's not needed here.

The first method necessary is getView(). This method creates a new View for each image added to the ImageAdapter. When this is called, a View is passed in, which is normally a recycled object (at least after this has been called once), so there's a check to see if the object is null. If it is null, an ImageView is instantiated and configured with desired properties for the image presentation:

setLayoutParams(ViewGroup.LayoutParams) sets the height and width for the View—this ensures that, no matter the size of the drawable, each image is resized and cropped to fit in these dimensions, as appropriate.

setScaleType(ImageView.ScaleType) declares that images should be cropped toward the center (if necessary).

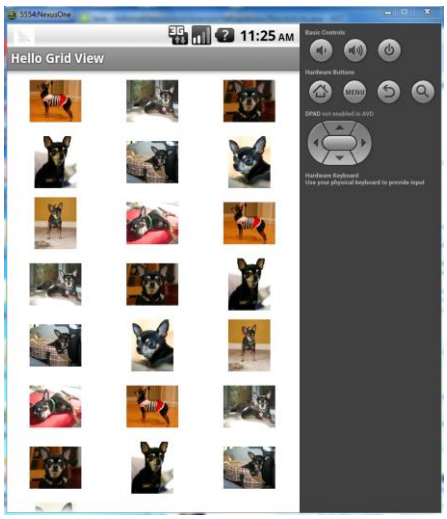
setPadding(int, int, int, int) defines the padding for all sides. (Note that, if the images have different aspect-ratios, then less padding will cause for more cropping of the image if it does not match the dimensions given to the ImageView.)

If the View passed to getView() is not null, then the local ImageView is initialized with the recycled View object.

At the end of the getView() method, the position integer passed into the method is used to select an image from the mThumbIds array, which is set as the image resource for the ImageView.

All that's left is to define the mThumbIds array of drawable resources.

6. Run the application. Your grid layout should look something like this:



CHECKPOINT 2 _____ *(Signed by Demonstrator)*

- Demonstrate the “Hello ListView” and “Hello GridView” programs.

References

- List View -

  http://developer.android.com/guide/topics/ui/layout/listview.html
- Grid View -

  http://developer.android.com/guide/topics/ui/layout/gridview.html

Spinner is a drop-down list like widget. An example can be found here:

- Spinner -

  http://developer.android.com/guide/topics/ui/controls/spinner.html

Hello WebKit

WebKit is a layout engine designed to allow web browsers to render web pages. The WebKit engine provides a set of classes to display web content in windows, and implements browser features such as following links when clicked by the user, managing a back-forward list, and managing a history of pages recently visited.

Web View

WebView allows you to create your own window for viewing web pages or even develop a complete browser. In this section, you'll create a simple Activity that can view and navigate web pages.

1. Create a new project named HelloWebView.
2. Open the res/layout/activity_main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout width="fill parent"
    android:layout_height="fill_parent"
/>
```

3. Now open the MainActivity.java file. At the top of the class, declare a WebView object. Then use the following code for the onCreate() method:

```
WebView mWebView;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity main);

    mWebView = (WebView) findViewById(R.id.webview);
    mWebView.getSettings().setJavaScriptEnabled(true);
    mWebView.loadUrl("http://www.google.com");
}
```

This initializes the member WebView with the one from the Activity layout; requests a WebSettings object with getSettings(); and enables JavaScript for the WebView with setJavaScriptEnabled(boolean). Finally, an initial web page is loaded with loadUrl(String).

4. Because this application needs access to the Internet, you need to add the appropriate permissions to the Android manifest file. Open the AndroidManifest.xml file and add the following as a child of the <manifest> element:

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

5. Now run the application.

You now have a simplest web page viewer. It's not quite a browser yet because as soon as it loads the link, the default Android Browser handles the Intent to view a web page (you can see the application title of HelloWebView disappears). This is because the default WebView isn't enabled to do so. Instead you can assign a WebViewClient to the class and allow this Activity to handle its own URL requests.

6. Towards the end of the onCreate(Bundle) method, set an instance of the WebViewClient to the WebView:

```
mWebView.setWebViewClient(new WebViewClient());
```

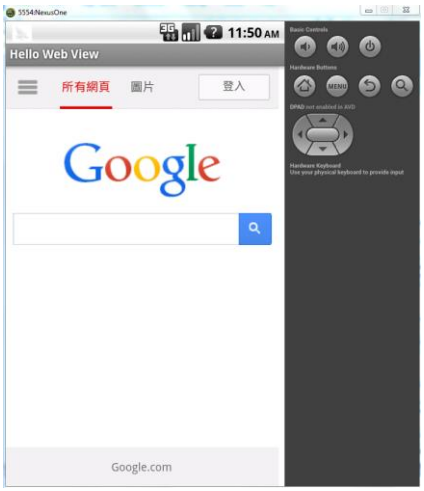
This creates a WebViewClient that will load any URL selected from this WebView into the same WebView. If you run the application again, new pages will now load in this Activity. However, you can't navigate back to previous pages. To do this, you need to handle the BACK button on the device, so that it will return to the previous page, rather than exit the application.

7. To handle the BACK button key press, add the following method inside the MainActivity.java:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_BACK) && mWebView.canGoBack()) {
        mWebView.goBack();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

This onKeyDown(int, KeyEvent) callback method will be called anytime a button is pressed while in the Activity. The condition inside uses the KeyEvent to check whether the key pressed is the BACK button and whether the WebView is actually capable of navigating back (if it has a history). If both are true, then the goBack() method is called, which will navigate back one step in the WebView history.Returning true indicates that the event has been handled. If this condition is not met, then the event is sent back to the system.

8. Run the application again. You'll now be able to follow links and navigate back through the page history. It should look like this:



9. If you want to remove the title bar to give some more space for web pages, you can set the "NoTitleBar" theme in the AndroidManifest.xml file:

```
android:theme="@android:style/Theme.NoTitleBar"
```

References

- Web View - http://developer.android.com/guide/webapps/webview.html

With Google's API, you can easily incorporate the Google Map in your application and provide location-based services through GPS. An example can be found here:

- Google Map View - https://developers.google.com/maps/documentation/android/v1/hello-mapview

Application Design

A central feature of Android is that one application can make use of elements of other applications (provided those applications permit it). For example, if your application needs to display a scrolling list of images and another application has developed a suitable scroller and made it available to others, you can call upon that scroller to do the work, rather than develop your own. Your application doesn't incorporate the code of the other application or link to it. Rather, it simply starts up that piece of the other application when the need arises.

For this to work, the system must be able to start an application process when any part of it is needed, and instantiate the Java objects for that part. Therefore, unlike applications on most other systems, Android applications don't have a single entry point for everything in the application (no main() function, for example). Rather, they have essential components that the system can instantiate and run as needed. There are four types of components:

1. **Activity** – A component presents a visual user interface for one focused endeavor the user can undertake. For example, show a list of menu items users can choose from.
2. **Service** – A background component (no visual user interface) runs for an indefinite period of time. For example, play music in the backgound, or fetch data over the network as the user attends to other matters.
3. **Broadcast Receiver** - A component receives and reacts to broadcast announcements. For example, react on the change of time zone or battery level.
4. **Content Provider** – A component makes a specific set of the application's data available to other applications. For example, look up phone numbers from the system Contacts application.

Apart from content provider, the other three types of components are activated by asynchronous messages called **Intent**, which is a description of an operation to be performed.

BMI Calculator

The body mass index (BMI) is a statistical measure of body weight based on a person's weight and height. It is used to estimate a healthy body weight based on a person's height.

$$\mathrm{BMI} = \frac{\mathrm{mass\ (kg)}}{(\mathrm{height(m)})^2}$$

In this section, you will create a multi-activity application that lets the user enter his/her height and weight and then report his/her BMI value with an advice. This example demonstrates how to use intents to link up two components in an application.

1. Create a new project named BMI.
2. Save the following sample image files into the project's res/drawable-mdpi/ directory.
   - bot_*.png
   - icon_128.png
3. Define some external values in res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BMI</string>
    <string name="bmi_height">Height (cm)</string>
    <string name="bmi_weight">Weight (kg)</string>
    <string name="bmi_btn">See BMI Report Now</string>
    <string name="bmi_result">Your BMI value is</string>
    <string name="bmi_warning">Height/Weight cannot be empty.</string>
    <string name="advice_light">You need more calories!</string>
    <string name="advice_average">You look great!</string>
    <string name="advice_heavy">You should be on diet!</string>
    <color name="bgcolor">#005990</color>
</resources>
```

There are two types of values – string and color. They will be referenced in your application where the bgcolor is the background color value.

4. Open the res/layout/activity_main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/bgcolor">

    <ImageView android:id="@+id/image"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_gravity="center_horizontal"
        android:paddingTop="40dp"
        android:paddingBottom="10dp"
        android:src="@drawable/icon_128" />

    <TextView android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/bmi_height" />
    <EditText android:id="@+id/height"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:text=""
        android:layout_marginBottom="10dp" />

    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/bmi_weight" />
    <EditText android:id="@+id/weight"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:text=""
        android:layout marginBottom="10dp" />

    <Button android:id="@+id/submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bmi_btn"
         android:onClick="calcBMI" />
</LinearLayout>
```

This XML defines the first UI in your application. It mainly displays two text boxes and a button for collecting user's height and weight.

5. Create another XML file named activity_report.xml and save it inside the res/layout/ folder. Insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout height="fill parent"
  android:background="@color/bgcolor"
  android:orientation="vertical">

   <ImageView android:layout_width="240dp"
    android:layout_height="240dp"
    android:layout_gravity="center_horizontal"
    android:paddingTop="40dp"
    android:paddingBottom="10dp"
    android:src="@drawable/bot_fit" android:id="@+id/report_image"/>

  <TextView android:layout width="wrap content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:paddingTop="10dp"
    android:paddingBottom="10dp"
    android:textSize="20dp"
    android:text="@string/bmi result" android:id="@+id/report result"/>

  <TextView android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:paddingTop="10dp"
    android:paddingBottom="10dp"
    android:textSize="20dp"
    android:text="@string/advice_average" android:id="@+id/report_advice"/>

</LinearLayout>
```

This XML is the second UI in your application. It mainly displays the calculated BMI value based on user's input and show an advice.

6. Open MainActivity.java and insert the following code for the MainActivity class:

```
public class MainActivity extends Activity {

    EditText vHeight;
    EditText vWeight;
    Button vButton;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // get views
        vHeight = (EditText) findViewById(R.id.height);
        vWeight = (EditText) findViewById(R.id.weight);
        vButton = (Button) findViewById(R.id.submit);

    }

            public void calcBMI(View v) {

            String height = vHeight.getText().toString();
            String weight = vWeight.getText().toString();

            Intent intent = new Intent(this, ReportActivity.class);
            Bundle bundle = new Bundle();
            bundle.putString("height", height);
            bundle.putString("weight", weight);
            intent.putExtras(bundle);
            startActivity(intent);
        }
    };
}
```

This activity displays the activity_main.xml layout file. When the submit button is clicked, the height and weight are obtained from the text boxes and are packed as a Bundle object, which can be used to hold any number of key-value pairs. It is then passed together with the intent to invoke the Report Activity class.

7. Create another class named ReportActivity.java that extends Activity:

```
public class ReportActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_report);

        Bundle bundle = getIntent().getExtras();

        double height = Double.parseDouble(bundle.getString("height"))/100;
        double weight = Double.parseDouble(bundle.getString("weight"));
        double bmi = weight / (height * height);

        DecimalFormat nf = new DecimalFormat("0.00");
        TextView result = (TextView) findViewById(R.id.report_result);
        result.setText(getString(R.string.bmi_result)+ " " + nf.format(bmi));

        // Give health advice
        ImageView image = (ImageView) findViewById(R.id.report_image);
        TextView advice = (TextView) findViewById(R.id.report_advice);
        if (bmi > 25) {
            image.setImageResource(R.drawable.bot_fat);
            advice.setText(R.string.advice_heavy);
        } else if (bmi < 20) {
```

```
            image.setImageResource(R.drawable.bot_thin);
            advice.setText(R.string.advice_light);
        } else {
            image.setImageResource(R.drawable.bot_fit);
            advice.setText(R.string.advice_average);
        }
    }
}
```

The activity displays the activity_report.xml layout file. The intent used to invoke this class can be retrieved by the getIntend() method, and so you can obtain the bundled key-value pairs (height and weight). It then calculates the BMI values and set the corresponding symbolic image and advice.

8. To use a custom icon for your application, open the AndroidManifest.xml file and update the drawable resource of the <application> element:

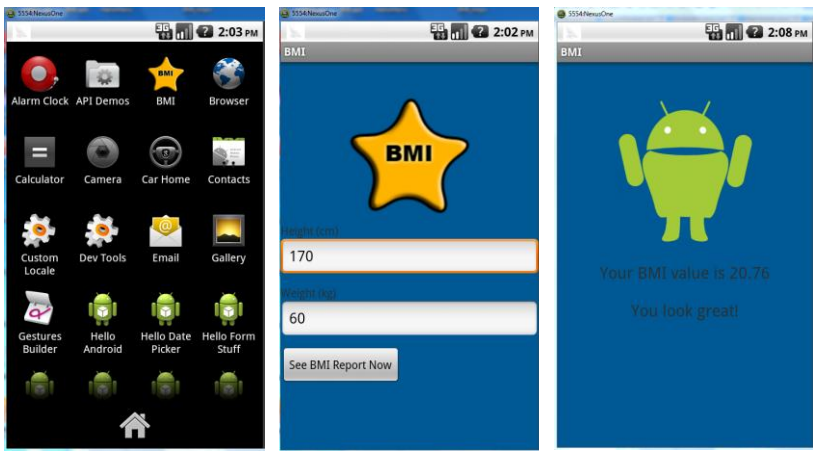
```
<application android:icon="@drawable/icon_128">
```

If you create an activity together with your project, Eclipse automatically adds this activity to your manifest. Activities that are not declared in the AndroidManifest.xml are invisible to the android system, and so cannot be instantiated. Therefore, you are required to add any newly created activities manually.

9. Insert the following code between the <application> tag in the manifest.

```
<activity android:name="com.example.bmi.ReportActivity"
android:label="@string/app_name">
```

10. Run the application. It should look like this:



Data Validation

You now have a simple BMI calculator that can measure your body weight. However a user may be able to submit empty values for calculation and it will cause an exception (force close) in your program.

1. Open the MainActivity.java and insert the following code for the onClick() method:

```
    public void CalcBMI(View v) {

        String height = vHeight.getText().toString();
        String weight = vWeight.getText().toString();

        if (height.equals("") || weight.equals("")) {
            Toast.makeText(MainActivity.this, R.string.bmi_warning,
                    Toast.LENGTH_LONG).show();
        } else {
            Intent intent = new Intent(this, ReportActivity.class);
            Bundle bundle = new Bundle();
            bundle.putString("height", height);
            bundle.putString("weight", weight);
            intent.putExtras(bundle);
            startActivity(intent);
        }
    }
```

The handler now will check on the inputs and use the handy Toast class to display a warning message.

2. Run the application again. It should work properly now even no input.

User Preferences

Being a user, you may not want to enter your height and weight every time, especially your height and weight won't change very dramatically. You can actually save user's data in a persistent storage so that you can get it back when the application is started next time.

1. Open MainActivity.java and insert the following two methods to the class:

```
public void savePreferences(String h, String w) {
    SharedPreferences pref = getSharedPreferences("BMI", MODE_PRIVATE);
    pref.edit().putString("height", h).commit();
    pref.edit().putString("weight", w).commit();
}

public void loadPreferences() {
    SharedPreferences pref = getSharedPreferences("BMI", MODE_PRIVATE);
    vHeight.setText(pref.getString("height", "0"));
    vWeight.setText(pref.getString("weight", "0"));
}
```

The “BMI” string is a name used to identify the save. You may use any other name as you like. The MODE_PRIVATE value is used to restrict access from any other applications of different user id (user id of linux).

2. Insert this code at the end of the **onCreate()** method:

```
loadPreferences();
```

3. Similarly, insert this code at the end of the **calcBMI()** method:

```
savePreferences(height,weight);
```

4. Run the application again. Now your application will load the height and weight when it starts and save them when the submit button is clicked.

Landscape Support

Unlike desktop applications, mobile applications usually have two UIs for portrait and landscape orientation. You can press **Ctrl-F12** to rotate the screen of your emulator. You will find that the BMI layouts defined previously are too big to fit into the screen. In Android, you can define multiple sets of resources that the system will automatically adapt to when configuration is changed.

1. Make two copies of the res/layout folder. Rename the folders as below:
   - res/layout-land (alternative resource for landscape mode)
   - res/layout-port (alternative resource for portrait mode)

The original res/layout folder remains as the default resource when no alternatives can be matched.

2. Create the res/layout-land/activity_main.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/bgcolor">

    <ImageView android:id="@+id/image"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerVertical="true"
        android:paddingTop="20dp"
        android:paddingBottom="20dp"
        android:src="@drawable/icon_128" />

    <TextView android:id="@+id/height_text"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/image"
        android:layout_alignTop="@id/image"
        android:text="@string/bmi_height" />
    <EditText android:id="@+id/height"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/height_text"
        android:layout_alignLeft="@id/height_text"
        android:numeric="integer"
        android:text=""
        android:layout_marginBottom="10dp" />

    <TextView android:id="@+id/width_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/height"
        android:layout alignLeft="@id/height"
        android:text="@string/bmi weight" />
    <EditText android:id="@+id/weight"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/width_text"
        android:layout alignLeft="@id/width text"
        android:numeric="integer"
        android:text=""
        android:layout_marginBottom="10dp" />

    <Button android:id="@+id/submit"
        android:layout width="wrap content"
        android:layout_height="wrap_content"
        android:layout_below="@id/weight"
        android:layout_alignLeft="@id/weight"
        android:text="@string/bmi_btn"
         android:onClick="calcBMI" />

</RelativeLayout>
```

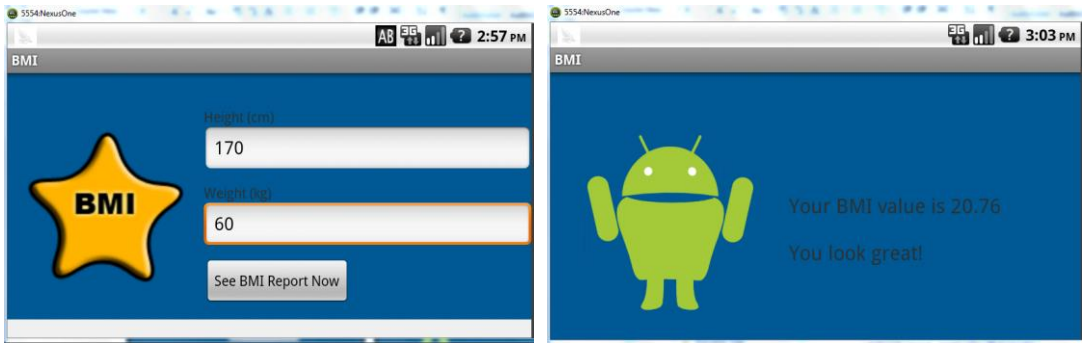
3. Similarly, create the res/layout-land/activity_report.xml file and insert the following:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/bgcolor">

    <ImageView android:id="@+id/report_image"
        android:layout_width="240dp"
        android:layout_height="240dp"
        android:layout_centerVertical="true"
        android:paddingTop="40dp"
        android:paddingBottom="10dp"
        android:src="@drawable/bot_fit" />

        <TextView android:id="@+id/report_result"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerVertical="true"
            android:layout_toRightOf="@id/report_image"
            android:paddingTop="10dp"
            android:paddingBottom="10dp"
            android:textSize="20dp"
            android:text="@string/bmi_result" />

        <TextView android:id="@+id/report_advice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout below="@id/report result"
            android:layout alignLeft="@id/report result"
            android:paddingTop="10dp"
            android:paddingBottom="10dp"
            android:textSize="20dp"
            android:text="@string/advice_average" />
```

```
</RelativeLayout>
```

These two layout files use RelativeLayout to arrange its children views in a wide screen.

4. Run the application again. You should see something like this:



CHECKPOINT 3 _____ *(Signed by Demonstrator)*

- Demonstrate your BMI calculator. Show that the app's UI can adapt to different orientations.

References

To see how to apply localization for different languages and device configurations, see the documentation and example here:

- Localization - http://developer.android.com/guide/topics/resources/localization.html
- Support Different Languages - http://developer.android.com/training/basics/supporting-devices/languages.html

Conclusion

Congratulation! You are ready to start your Android application project now! Bear in mind that mobile application design is all about user. A good application is not necessary to be technically complicated, but can always enrich user experience.

User-centered design is a design philosophy in which the needs, wants, and limitations of end users of a product are given extensive attention at each stage of the design process. Being a product designer, you are required to analyze and foresee how users are likely to use a product, and to test the validity of your assumptions with regards to user behavior in real world tests with actual users.

When you start to plan your application, try to discuss these questions among your group:

- Who are the users of the application?
- What are the users' tasks and goals?
- What are the users' experience levels with the application?
- What functions do the users need from the application?
- What information might the users need, and in what form do they need it?
- How do users think the application should work?

-END-