Running and Debugging Your Android Applications

You've created your fi rst project and created the run and debug confi gurations for it. Before making any changes, test your installation and confi gurations by running and debugging the Hello World project.

From the **Run** menu, select **Run** or **Debug** to launch the most recently selected confi guration, or select **Open Run Dialog** ... or **Open Debug Dialog** ... to select a confi guration to use.

If you're using the ADT plug-in, running or debugging your application:

- ❑ Compiles the current project and converts it to an Android executable (.dex).
- ❑ Packages the executable and external resources into an Android package (.apk).
- ❑ Starts the emulator (if it's not already running).
- ❑ Installs your application onto the emulator.
- ❑ Starts your application.

If you're debugging, the Eclipse debugger will then be attached, allowing you to set breakpoints and debug your code.
If everything is working correctly, you'll see a new Activity running in the emulator, as shown in Figure 2-6.

Figure 2-6

Understanding Hello World

With that confi rmed, let's take a step back and have a real look at your fi rst Android application. Activity is the base class for the visual, interactive components of your application; it is roughly equivalent to a Form in traditional desktop development. The following snippet shows the skeleton
code for an Activity-based class; note that it extends Activity, overriding the onCreate method.

```
package com.paad.helloworld;
import android.app.Activity;
import android.os.Bundle;
public class HelloWorld extends Activity {

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
```

```
}
}
```

What's missing from this template is the layout of the visual interface. In Android, visual components are called *Views*, which are similar to controls in traditional desktop development.

In the Hello World template created by the wizard, the onCreate method is overridden to call setContentView, which lays out the user interface by infl ating a layout resource, as highlighted below:

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
}
```

The resources for an Android project are stored in the res folder of your project hierarchy, which includes drawable, layout, and values subfolders. The ADT plug-in interprets these XML resources to provide design time access to them through the R variable as described in Chapter 3. The following code snippet shows the UI layout defi ned in the main.xml fi le created by the Android project template:

```
<?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">
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Hello World, HelloWorld"
/>
</LinearLayout>
```

Defi ning your UI in XML and infl ating it is the preferred way of implementing your user interfaces, as it neatly decouples your application logic from your UI design.

To get access to your UI elements in code, you add identifi er attributes to them in the XML defi nition. You can then use the findViewById method to return a reference to each named item. The following XML snippet shows an ID attribute added to the TextView widget in the Hello World template:

```
<TextView
android:id="@+id/myTextView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Hello World, HelloWorld"
/>
```

And the following snippet shows how to get access to it in code:

```
TextView myTextView = (TextView)findViewById(R.id.myTextView);
```

Alternatively (although it's not considered good practice), if you need to, you can create your layout

directly in code as shown below:

```
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
LinearLayout.LayoutParams lp;
lp = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
LinearLayout.LayoutParams textViewLP;
```

```
textViewLP = new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT, LayoutParams.WRAP_CONTENT);

LinearLayout ll = new LinearLayout(this);

ll.setOrientation(LinearLayout.VERTICAL);

TextView myTextView = new TextView(this);

myTextView.setText("Hello World, HelloWorld");

ll.addView(myTextView, textViewLP);

this.addContentView(ll, lp);

}
```

All the properties available in code can be set with attributes in the XML layout. As well as allowing easier substitution of layout designs and individual UI elements, keeping the visual design decoupled from the application code helps keep the code more concise.

*The Android web site (*http://code.google.com/android/documentation.html*) includes several excellent step-by-step guides that demonstrate many of the features and good practices you will be using as an Android developer. They're easy to follow and give a good idea of how Android applications fi t together.*