Resources

Introduction

- Resource is a static collection of bits held outside the Java source code.
- Examples for resources include files (e.g. image file) or values (e.g. strings used in our application) that are bounded to an executable application.

  When changing the bits of a specific resource there is no need to recompile the source code. The way we refer that resource remains unchanged.

Introduction

- We define the resources in XML files. We can reuse them in other resources definitions as well as in our source code.
- The resources are managed by the `R.java` source file. It manages their generation and usage.

String Resources

- String resources are the simplest form of resources.
- The XML files that include the definitions of string resources should be located within the '`res/values/`' directory.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">hello</string>
    <string name="app_name">hello friends</string>
</resources>
```

- When editing any of the resources files the ADT plug-in automatically updates the R.java source code file.

String Resources

- When a string contains a quotation mark (“) or apostrophe (') we must escape it by preceding it with a backslash.

```
...
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">hello</string>
    <string name="app_name">i love \'canada\'</string>
</resources>
...
```

String Resources

- We can use the string resources from within the XML layout file referring them using the following syntax "`@string/____`".

```
...
<TextView   android:text="@string/hello"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></TextView>
...
```

String Resources

- We can use the string resources from within the Java source file by calling the `getString()` method passing over the ID integer value of the string resource we are interested at.
- We refer that ID integer number by referring the R auto generated class specifying the int final static variable that holds the ID number of the resource we are interested at.

```
...
String str = getResources().getString(R.string.hello);
...
```

String Resources

- The android platform supports strings formats. The string resource can contain a placeholder(s) for holding data that will be set during the execution of our code.
- Assuming the string value is "`my name is %1$s`" we will be able to replace the string placeholder with a specific value set during the execution of our code.

```
...
String str = getString(R.string.txt);
String result = String.format(str,"Haim");
...
```

String Resources

- We can include within the string resource simple HTML tags, such as <b>, <i> and <u> in order to format the texts in accordance with our needs.

```
<resources>
    <string name=”txt”>the <b>important</b> man</string>
</resources>
```

String Resources

```
public class StringsResources extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Resources res = getResources();
        String str = res.getString(R.string.text_on_button);
        Button bt = new Button(this);
        bt.setText(str);  //bt.setText(R.string.text_on_button);
        LinearLayout layout = new LinearLayout(this);
        layout.addView(bt);
        setContentView(layout);
    }
}
```

String Resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, StringsResources!</string>
    <string name="app_name">strings resources sample</string>
    <string name="text_on_button">press here!</string>
</resources>
```

String Resources

Arrays Resources

- We can create array resources by using the `<array>` element.
- The XML file can be named with any name we choose. It should be saved within the 'values' folder.

Arrays Resources

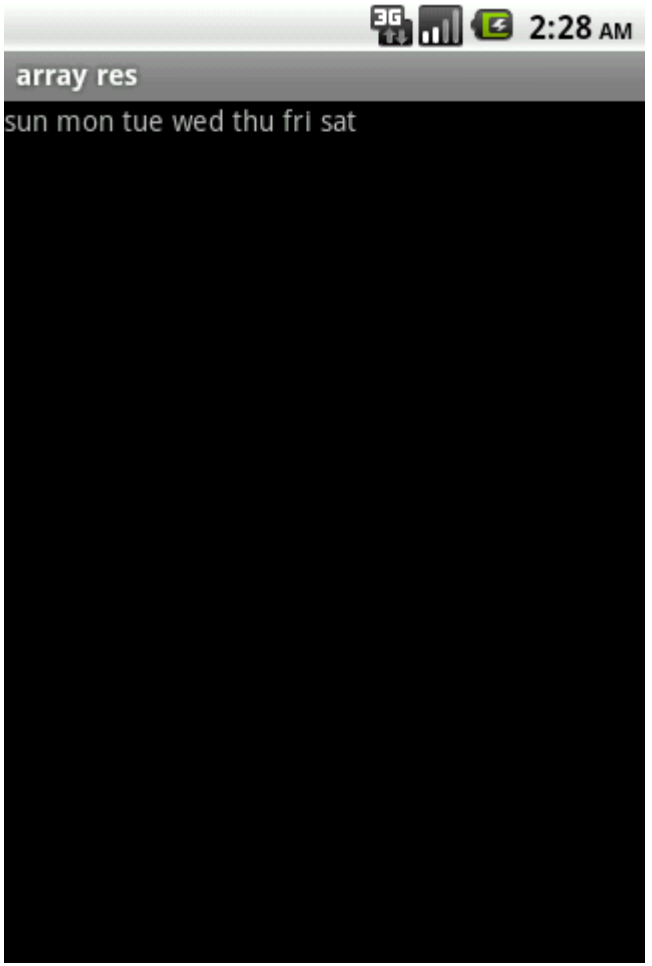
```
package com.abelski.samples;

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

public class ArrayResourceDemoActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        String[] vec = getResources().getStringArray(R.array.days);
        StringBuffer sb = new StringBuffer();
        for(String str : vec)
        {
            sb.append(str);
            sb.append(" ");
        }
        TextView tv = new TextView(this);
        tv.setText(sb.toString());
        setContentView(tv);
    }
}
```

Arrays Resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <array name="days">
        <item>sun</item>
        <item>mon</item>
        <item>tue</item>
        <item>wed</item>
        <item>thu</item>
        <item>fri</item>
        <item>sat</item>
    </array>
    <array name="months">
        <item>jan</item>
        <item>feb</item>
        <item>mar</item>
        <item>apr</item>
    </array>
</resources>
```

we can name this file with any name we choose... the file should be saved within the 'values' folder within 'res'.

Arrays Resources



Color Resources

- We can define colors as resources by using the `<color>` xml element within the XML file we save within the 'values' folder, sub folder of 'res'.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="alert_color">#ff33cc</color>
    <color name="nice_color">#aaff3322</color>
</resources>
```

we can name the file whatever we want... the file must be saved within the 'values' folder, sub folder within 'res'.

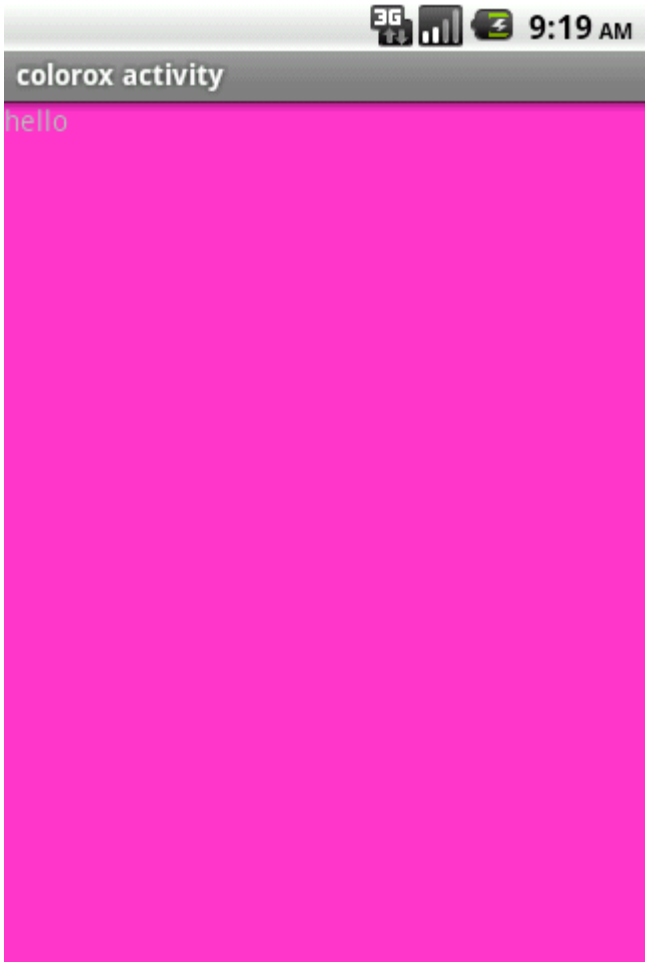
Color Resources

```
package com.abelski.samples;

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

public class ColoroxActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setBackgroundColor(getResources().
            getColor(R.color.alert_color));
        tv.setText("hello");
        setContentView(tv);
    }
}
```

Color Resources

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="alert_color">#ff33cc</color>
    <color name="nice_color">#aaff3322</color>
</resources>
```

Color Resources



Drawable Resources

- The android platform supports using images of the following formats: `PNG`, `JPG` and `GIF`. The `PNG` format is the preferred one.
- We can use images wherever we can use a Drawable resource, such as the background of a specific image button.

Drawable Resources

- Within XML layout files we can refer an images as a `@drawable` resource using the '`@drawabe/___`' syntax.

```
...
<ImageButton        android:id="@+id/ImageButton01"
                    android:background="@drawable/toto"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content">
</ImageButton>
...
```

- The base name of the image file is the ID name we use when referring it from within the XML layout file.

Drawable Resources

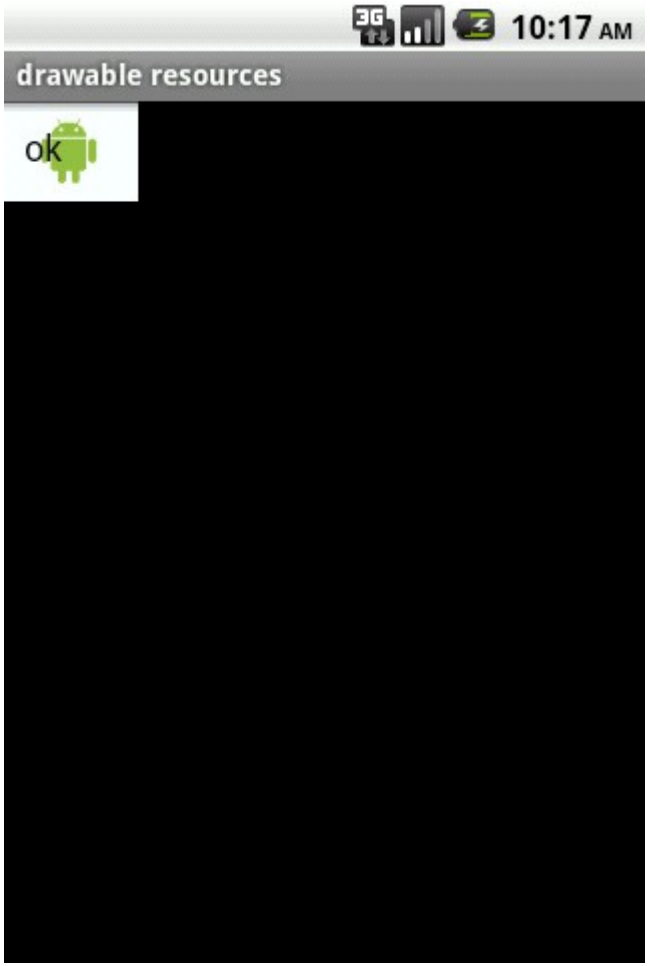
- Within the Java source code file, wherever we need an image resource ID we should use `R.drawable` together with the base name of image file.

```
...
ImageButton bt = new ImageButton(this);
bt.setBackgroundResource(R.drawable.toto);
...
```

Drawable Resources

```
public class DrawableResourcesActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        EditText editText = new EditText(this);
        editText.setBackgroundResource(R.drawable.andy);
        layout.addView(editText);
        setContentView(layout);
    }
}
```

Drawable Resources



XML Resources

- XML resources are XML files we place within the `res/xml/` folder.
- We can access these resources by calling the `getXml()` method on the `Resources` object passing over the ID of the specific XML resource we want to use.

```
...
XmlPullParser parser = getResources().getXml(R.xml.words);
...
```

- The `XmlPullParser` is an `XML` event driven parser.

XML Resources

- We can easily access XML files that were stored within the `res/xml` folder. These files are compiled into an efficient binary format when deployed.

```
...
Resources resources = this.getResources();
XmlPullParser parser = resources.getXml(R.xml.cars);
...
```

XML Resources

```
public class SimpleCountriesListActivity extends ListActivity
{
    ArrayList<String> list = new ArrayList<String>();

    @Override
    public void onCreate(Bundle bndl)
    {
        super.onCreate(bndl);
        try
        {
            XmlPullParser parser = getResources().getXml(R.xml.countries);
            int eventType = parser.getEventType();
            while(eventType!=XmlPullParser.END_DOCUMENT)
            {
                if(eventType == XmlPullParser.TEXT)
                {
                    list.add(parser.getText());
                }
                eventType = parser.next();
            }
        }
```

XML Resources

```
        catch (Exception e)
        {
            Log.i("xmlxml",e.getMessage());
        }
        setListAdapter(new ArrayAdapter<String>(
            this,
            android.R.layout.simple_list_item_1,
            list));
    }

    public void onListItemClick(
        ListView parent, View v, int position, long id)
    {
        String toastText = "i like "+list.get(position);
        Toast toast = Toast.makeText(this,toastText,Toast.LENGTH_SHORT);
        toast.show();
    }
}
```

XML Resources

```
<?xml version="1.0" encoding="UTF-8"?>
<countries>
    <country>italy</country>
    <country>russia</country>
    <country>france</country>
    <country>canada</country>
</countries>
```

XML Resources

Layout Resources

- Layout resources are responsible for the layout of the user interface. Each layout resource is defined in a separated XML file located within '`/res/layout`' folder.
- Each layout resource `XML` file is mapped with a specific `ID` within the `R.java` source code file.

Layout Resources

```
package com.abelski.samples;

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

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

Layout Resources

```
<?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:id="@+id/text_a"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

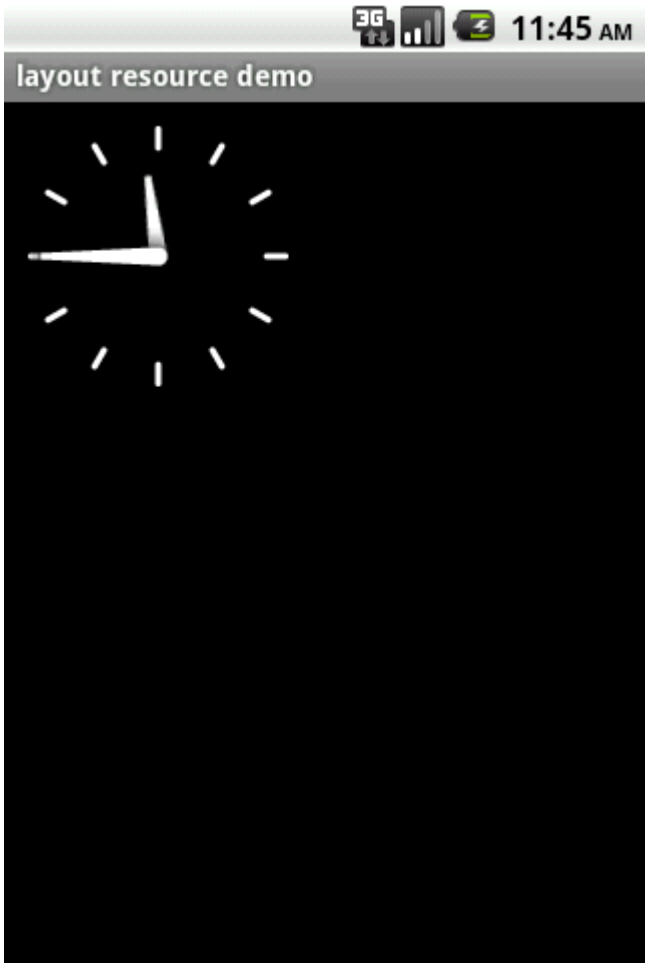
main.xml

Layout Resources

```
<?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">

    <AnalogClock android:id="@+id/AnalogClock01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></AnalogClock>

</LinearLayout>
```

myothermain.xml

Layout Resources



Layout Resources

- For each layout XML file we will find within the declaration of the `layout` static inner class (within '`R.java`') a declaration of a final static int variable that its name is identical to the name of the XML layout file.

Layout Resources

```
package com.abelski.samples;

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 AnalogClock01 = 0x7f050001;
        public static final int text_a = 0x7f050000;
    }
```

Layout Resources

```
    public static final class layout
    {
        public static final int main = 0x7f030000;
        public static final int myothermain = 0x7f030001;
    }

    public static final class string
    {
        public static final int app_name = 0x7f040001;
        public static final int hello = 0x7f040000;
    }
}
```

Dimension Resources

- The dimension is specified with a number followed by its unit of measure (e.g. 10px, 2in, 5dp stc.).
- We define the dimension resource using XML file we save inside the 'values' folder (within the 'res' folder).
- We use the `<dimen>` XML element in order to define the dimension resource.

Dimension Resource

```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="text_size">46dp</dimen>
    <dimen name="my_font_size">22dp</dimen>
</resources>
```

dimens.xml

Dimension Resource

activity_my.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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity">
    <TextView
        android:text="@string/hello_world"
        android:textSize="@dimen/my_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Dimension Resource

Color State List Resource

- The `ColorStateList` is a class. Each object instantiated from this class represents the various color states of a specific view. Each object instantiated from this class actually maps the possible states of a specific View object with specific colors.

Color State List Resource

- Using a plain simple XML document we can indirectly instantiate this class and get an object we can later applies to specific view. The XML document should be saved within the 'color' folder inside 'res'.

Color State List Resource

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

<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true" android:color="@color/color2" />
    <item android:state_pressed="false" android:color="@color/color3" />
</selector>
```

my_bt_states.xml

Color State List Resource

my_bt_states.xml

```
<LinearLayout 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:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MyActivity">
```

Color State List Resource

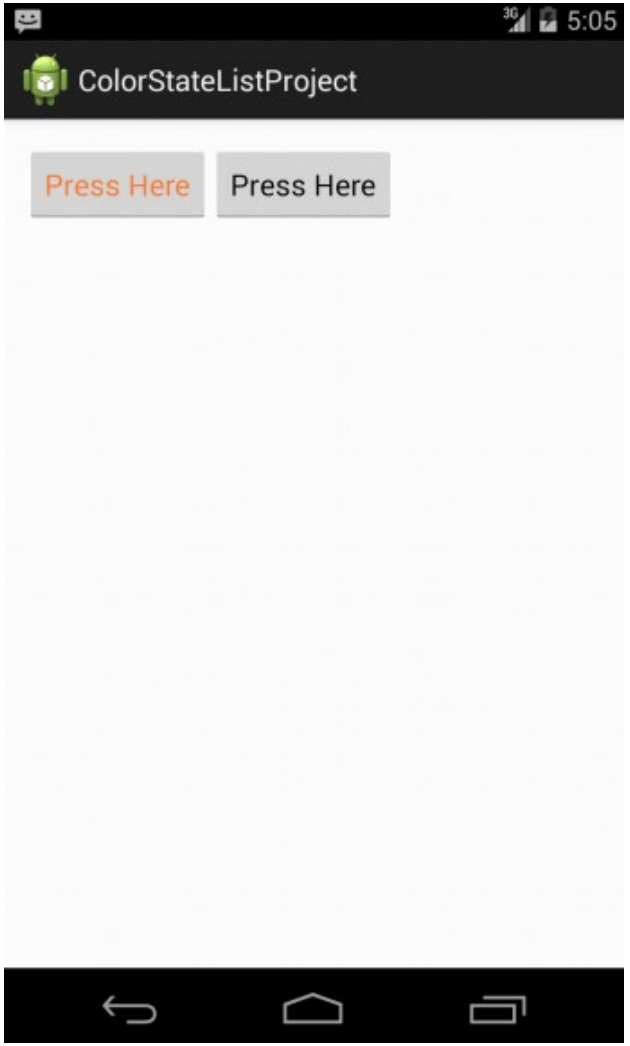
```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bt_text"
        android:id="@+id/button_a"
        android:layout_alignParentEnd="true"
        android:textColor="@color/my_bt_states"
          />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bt_text"
        android:id="@+id/button_b"
        android:layout_alignParentEnd="true"
          />

</LinearLayout>
```

Color State List Resource

```
public class MyActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
    }

}
```

MyActivity.java

Color State List Resource



Boolean Resources

- The boolean resource is a boolean value defined in XML file within the values folder.

Boolean Resources

```
<resources>
    <bool name="multiple_colors">true</bool>
    <bool name="melodic_sounds">true</bool>
    <bool name="secured_mode">true</bool>
</resources>
```

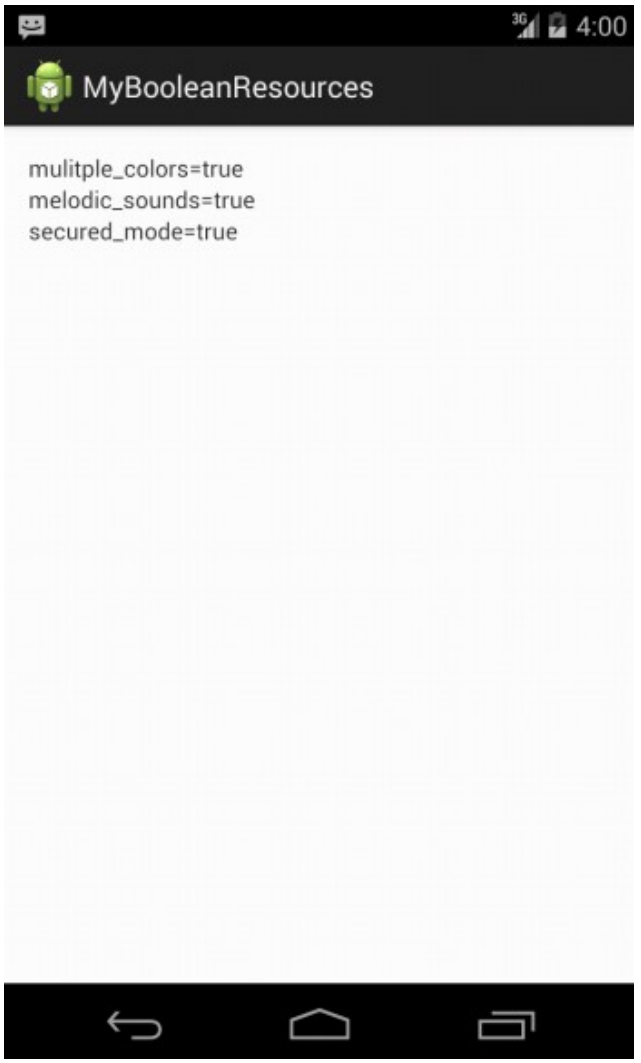
settings.xml

Boolean Resources

MyActivity.java

```
public class MyActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_my);
        TextView tf1 = (TextView)findViewById(R.id.tf1);
        TextView tf2 = (TextView)findViewById(R.id.tf2);
        TextView tf3 = (TextView)findViewById(R.id.tf3);
        Resources resources = getResources();
        tf1.setText("mulitple_colors="
           +resources.getBoolean(R.bool.multiple_colors));
        tf2.setText("melodic_sounds="
           +resources.getBoolean(R.bool.melodic_sounds));
        tf3.setText("secured_mode="
           +resources.getBoolean(R.bool.secured_mode));
    }
}
```

Boolean Resources



Resource Reference Syntax

- All resources of all types are referenced using an ID number.

Resource Reference Syntax

- When creating a new resource in our layout XML file it should be with the following syntax:

```
@+[package_name:]id/name
```

- The '`name`' is the name we choose to give the resource.
- Specifying the package is optional. If we do specify a package then the system will create a package for `R.java` file.

Resource Reference Syntax

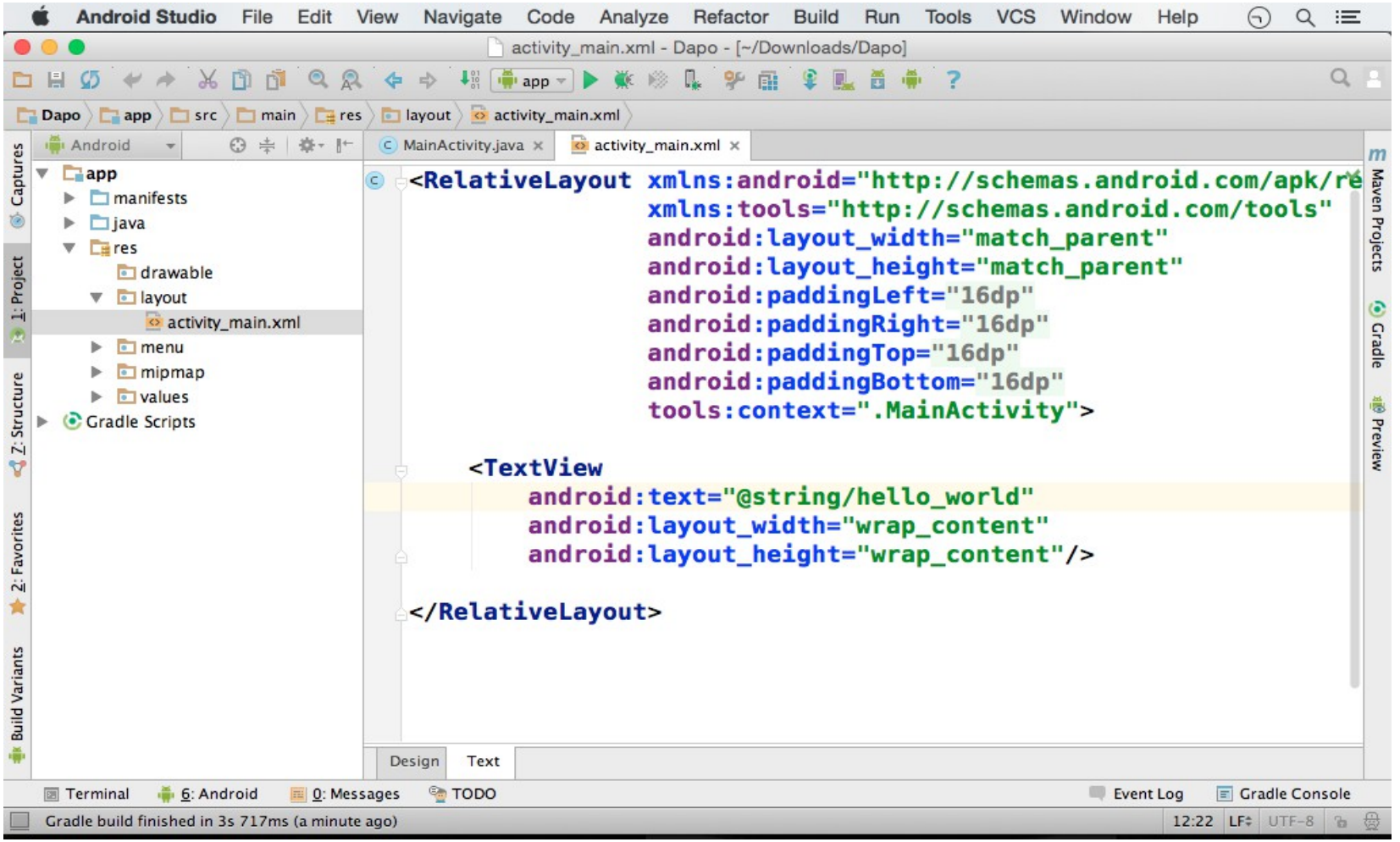
- When using a resource in our XML file we should use the following syntax:

```
@type/name
```

- The '`name`' is the name of the resource we want to use. The '`type`' is its type name (e.g. '`string`','`drawable`').

```
...
<ImageButton android:id="@+id/ImageButton01"
android:background="@drawable/icon"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></ImageButton>
...
```

Resource Reference Syntax



Resource Reference Syntax

- The type should be in according with one of the resource-type name spaces available in `R.java`:

```
R.drawable
R.id
R.layout
R.string
R.attr
R.style
...
```

Compiled Resources

- All resources placed within `/res` and its sub folders except for those places within the `/res/raw/` are compiled into the installable package.

Uncompiled Resources

- Placing the resource file within the `res/raw` directory we won't get it compiled into binary format. They will be included within the installable package 'AS IS'.

Uncompiled Resources

- Calling the `getResources()` method (defined within Activity) we can get a `Resources` object.
- We can call the `openRawResource()` method on a `Resources` object in order to get an input stream to the resource we want to work with.

```
...
Resources r = activity.getResources();
InputStream is = r.openRawResource(R.raw.test);
...
```

Resources Names

- Each resource has an ID number held in a static variable that belongs to one of the static inner classes in R.
- The name of that static variable is the very same name we specify as the name of the resource. It can be name of the string (string resource), the name of the image file (drawable resource), the name of the array (array resource), the name of the xml layout file (layout resource) etc.

Resources Names

- The name of the resource must obey the same rules that apply when creating a simple variable in Java.

Assets

- The files placed within the `assets` directory won't get a reference ID within the `R.java` file.
- Using the `AccessManager` class we can access these files.

```
...
AssetManager mnager = getAssets();
InputStream is = manager.open("filename.txt");
...
```

Internationalization & Localization

- Placing all texts outside the Java and the XML source code files is a good practice that helps with the localization of our application.

The `.apk` File

- The Android Asset Packaging Tool (AAPT) compiles all resources (except for the ones within 'raw') and places them all into the .apk file. This is the equivalent file to the known 'jar' file (in Java ME).
- The .apk file contains the android application's code and resources.
- The .apk file is the one gets installed onto the android device.