**Mobile App Tutorial**
**Deploying a Handler and Runnable for Timed Events**
**Creating a Counter**

**Description:**

Given that Android Java is event driven, any action or function call within an Activity Class must be called by an event. Up until now, we have used the Button attribute onClick or Listeners attached to SeekBars or EditText boxes to call events. If we need to call a cycle of events, such as displaying a series of images, initiating a countdown, or an animation / game loop, we need a means to have an Activity 'self call' a function repeatedly. We achieve this with a "Handler – Runnable" combination.

According to the Android Developer guide:

> A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

By sending messages into a queue, we can time these messages to exit the cue and call specific functions.

A Runnable is implemented by the Thread class and works with the Handler to called the function. The Runnable class has a method called 'run' that must be defined when the Runnable is created. The 'run' method will call the function we wish to repeat.

The Function / Handler / Runnable cycle can be summarized as follows:

1. External Function Call initiates the Sequence (from onCreate() or a user interaction).
2. Desired Action takes place via a function call.
3. A Check condition is measured within the function call.
4. Depending on Check condition, a Runnable is called with a Delay.
5. Runnable object Calls the desired Function and Sequence again.
6. When Check Condition is False – exit the sequence.

This Exercise will guide your through creating an App that:

1. Counts Up in whole numbers at one second intervals.
2. Features a Reset Button.

In later projects will apply the Handler / Runnable combination to control repeated events, animation loops, and game loops.

**Phase 1: Create the App Project**

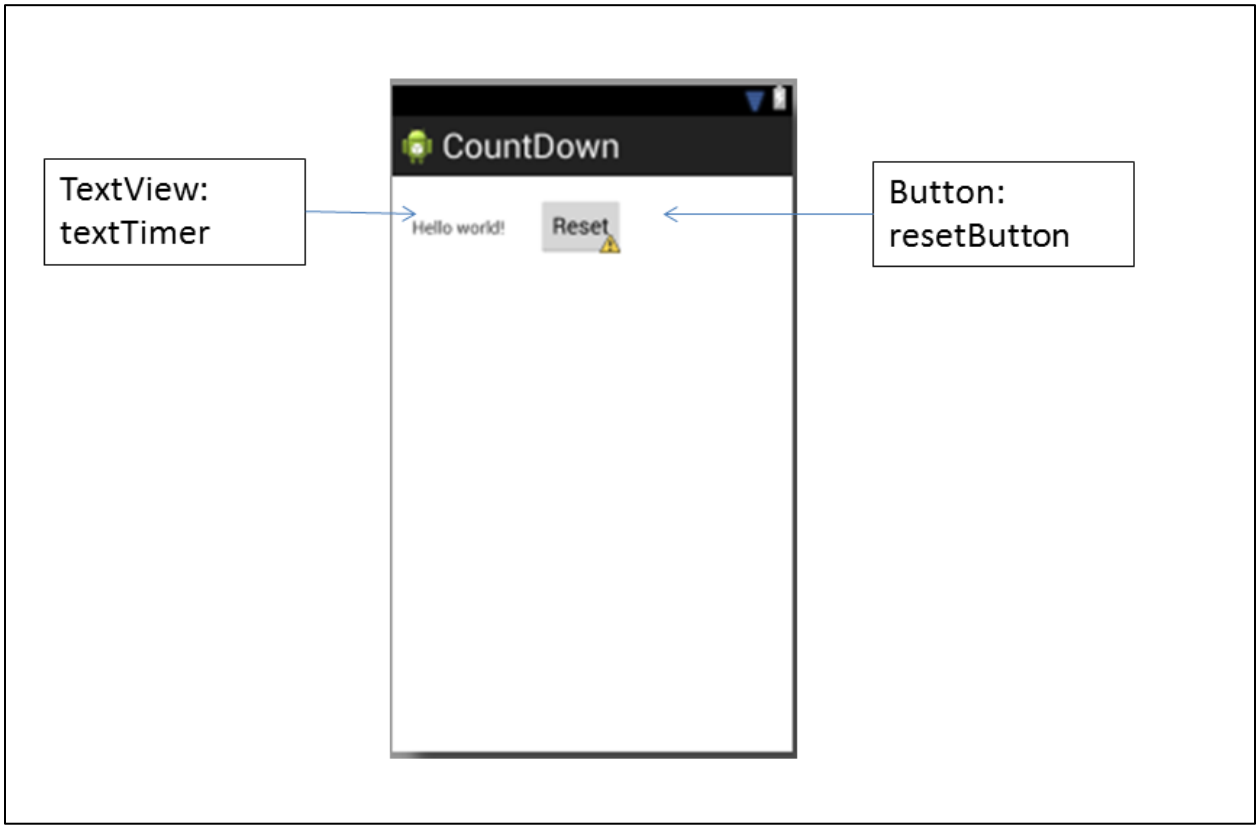
**Process:**

1. Start Eclipse and select new Project -> “Android Application Project”
2. Fill out the fields with the following:
    a. Application Name: CountDown
    b. Project Name: CountDown
    c. Package Name: com.example.countdown
3. Click Next
4. Click Next at the Configure Project Screen
5. Click Next at the Configure Launcher Screen
6. Click Next at the Create Activity Screen
7. Fill out the following fields in the New Blank Activity Screen
    a. Activity Name: Countdown
    b. LayoutName: activity_countdown
    c. Navigation Type: None
8. Click “Finish”

**Phase 2: Android XML User Interface Design**

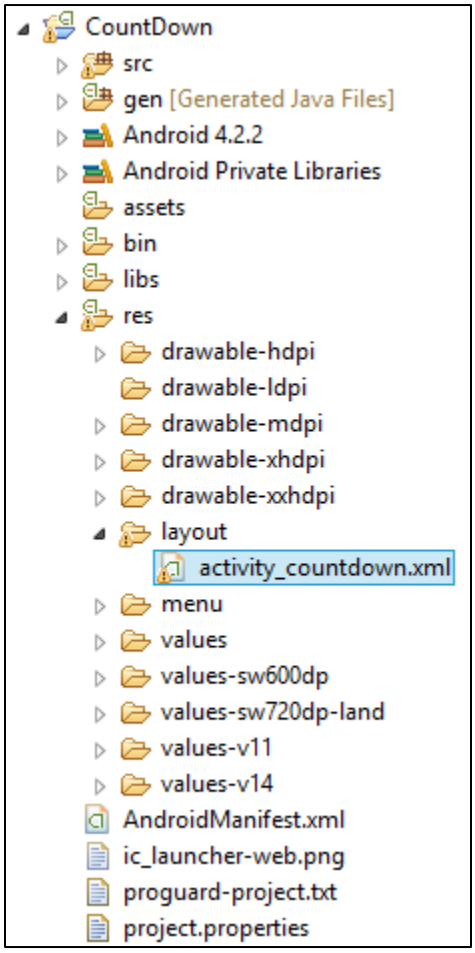
Because this is an exercise is only interested in experimenting with Handlers/Runnables, we will use a very basic XML Setup.

**Objects:**



**Process:**

1. Open the activity_countdown.xml file:



2. Modify the XML Code to setup the textTimer TextView object and the resetButton Button Object. You will also need to change the layout to Linear Layout.

```
<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:orientation="horizontal"
    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=".Countdown" >

    <TextView
            android:id="@+id/textTimer"
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:text="@string/hello_world" />

    <Button
        android:id="@+id/resetButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="reset"
        android:text="Reset" />

</LinearLayout>
```

**Phase 3: Write the Java Code for the Countdown.java class.**

**Process:**

1. Open the Countdown.java class and add the following fields. (Remember, when an object is not recognized by Eclipse, you can usually hover and import the needed Android or Java class.

```
 1 package com.example.countdown;
 2
 3 import android.os.Bundle;
 4 import android.os.Handler;
 5 import android.app.Activity;
 6 import android.view.Menu;
 7 import android.view.View;
 8 import android.widget.Button;
 9 import android.widget.TextView;
10
11 public class Countdown extends Activity {
12
13     // Fields
14     private TextView textTimer; // Displays the Text
15     private Button resetButton; // Used to reset the Counter
16
17     private int elapsedTime; // Used to hold the elapsed Time
18
19     private Handler h; // The Handler
20
21     private int RATE = 1000; // milliseconds - how fast to count
22
```

2. Modify the onCreate() method to bind the user interface objects to XML and initialize the variables and Handler. Note that on line 37, the count function is called by onCreate() to start the Handler / Runnable Sequence.

```
22
23      @Override
24      protected void onCreate(Bundle savedInstanceState) {
25          super.onCreate(savedInstanceState);
26          setContentView(R.layout.activity_countdown);
27
28          // Bind to XML
29          textTimer = (TextView)findViewById(R.id.textTimer);
30          resetButton = (Button)findViewById(R.id.resetButton);
31
32          elapsedTime = 0; // Set the variable for time to 0
33
34          // Initialize the Handler
35          h = new Handler();
36
37          count(); // Starts the count
38
39      } // end onCreate
40
```

3. Write the function for reset. This is called by the restButton's onClick attribute.

```
47
48      public void reset(View v) {
49          elapsedTime = 0;
50      }
51
```

4. Write the count() function to increase the count, display, and call the Runnable.

```
51
52      // Increases time counter and calls the Runnable r
53      public void count() {
54          elapsedTime++; // increase the counter
55          textTimer.setText(String.valueOf(elapsedTime)); // set the View
56          h.postDelayed(r, RATE); // Call the Runnable
57      } // end count()
58
```

5. Write the Runnable r to complete the sequence build:

```
58
59      // Runnable - calls the count() function to continue the sequence
60      private Runnable r = new Runnable() {
61          public void run() {
62              count(); // calls the count function
63          }
64      };  // End Runnable r
65
66  } // end class Countdown
67
```

6. Save and run the program on the emulator or device. The App should count up in seconds. (Yes, I know it is not that exciting . . . More will come later).

**Classwork Scoring: (Maximum 25 Points)**

This App is misnamed! It does not count down, it counts up. Please note the scoring for options to fix the App.

1. Complete the Directions and create a working CountDown App as described above: (20 Points)
2. Modify the App to Count Down from 60 and stop at 0. (2 points)
3. Modify the App to Include a Button to Pause and Restart the Count. (3 points)