About This Article

This article provides information about how to use JUnit testcases in Android applications to test the application. Android application developers can use this article as reference to implement JUnit testcases and test their applications.

Scope:

This article is intended for Android developers wishing to develop mobile applications. It assumes basic knowledge of Android and Java programming languages.

To find out more about Android, please refer to the Knowledge Base under Samsung Developers.
https://developer.samsung.com/android

Introduction

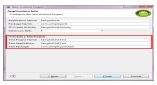
The Android testing framework (*android.test* package) is based on a popular JUnit framework (seehttp://www.junit.org/). The JUnit *junit.framework.TestCase* class is a main building block of the Junit framework.
This article assumes that the reader has some familiarity with Junit.See http://www.junit.org/ for information about JUnit.
In this article you will learn about how to build JUnit testcases and test your android application. Before beginning the walkthrough, it is important to point out JUnit and Android additions in the *android.test* package. Android has also added its extensions in the *android.test* package. All android test cases class inherits from*android.test.AndroidTestCase*. The main purpose of *AndroidTestCase* is to test android dependent objects.*AndroidTestCase* extends both the *TestCase* and *Assert* classes. It also offers Android specific setup, tear down and helper methods. The Android Test framework has component specific test classes, mainly related to Activity, Service and Content Provider testing.

The sample project illustrates the JUnit testcases for Activity Testing. The example converts kilos to pounds and vice versa. By using unit test cases we can check the output of the both conversions.

Walkthrough

Creating A Project

Create a **SampleJUnitTest** project along with **SampleJUnit** project to test. Check the "Create a Test Project" checkbox as shown below.



Click **Finish**, this creates two project folders for your application as shown below.

Now you have the *SampleJUnit* project and *SampleJUnitTest* test project. Next implement your test cases in*SampleJUnitTest*. Implement some UI in *SampleJUnit* project and write the test cases for it in *SampleJUnitTest*project.
The following screen shows the UI implementation of the scenario which converts the kilos to pounds and vice versa.

As shown above two edittexts, two buttons and one textview is available in the given UI. The task is to test that all the views are working properly or not using JUnit testcases. The following xml file shows the UI implementation.

main.xml

```
<?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" >
    <Button
        android:id="@+id/Pounds"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:text="Convert to Pounds" />
    <EditText
        android:id="@+id/inputvalueK"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/Pounds"
        android:hint="No. of Kilos" />
    <Button
        android:id="@+id/Kilos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@id/Pounds"
        android:text="Convert to Kilos" />
    <EditText
        android:id="@+id/inputvalueP"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/Pounds"
        android:layout_toLeftOf="@id/Kilos"
        android:hint="No. of Pounds" />
    <TextView
        android:id="@+id/resultview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/Kilos"
        android:text="Result  " />
```

```
</RelativeLayout>
```

This *SampleJUnit* is responsible for converting the given kilos to pounds and vice versa. Create one activity in*SampleJUnit* projct and name it *SampleJUnitActivity*. The following class file shows the corresponding activity.

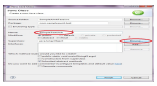
```
public class SampleJUnitActivity extends Activity implements OnClickListener{
/** Called when the activity is first created. */
Button mCKilos,mCPounds;
EditText mKilos,mPounds;
TextView mResult;
@Override
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      mResult = (TextView)findViewById(R.id.resultview);
      mKilos = (EditText) findViewById(R.id.inputvalueK);
      mPounds = (EditText) findViewById(R.id.inputvalueP);
      mCKilos = (Button) findViewById(R.id.Kilos);
      mCPounds = (Button) findViewById(R.id.Pounds);
      mCKilos.setOnClickListener(this);
      mCPounds.setOnClickListener(this);
}
@Override
public void onClick(View v) {
         // TODO Auto-generated method stub
         switch(v.getId())
         {
         case R.id.Kilos:
         {
              double pounds = Double.parseDouble(mPounds.getText().toString());
              double kilos = pounds * 0.45359237;
              mResult.setText(new Double(kilos).toString());
         }
         break;
         case R.id.Pounds:
         {
              double kilos = Double.parseDouble(mKilos.getText().toString());
              double pounds = kilos * 2.20462262;
              mResult.setText(new Double(pounds).toString());
         }
         break;
         }
}
}
```

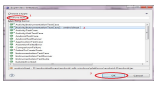
Now run the application and the following screen appears:

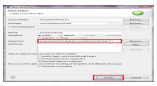
Once the UI implementation is done, the next task is to write the testcases in *SampleJUnitTest* corresponding to the*SampleJUnit*.

Creating a Testcase class

Create a testcase class in a test project and name it *SampleTestUnit*, which is a subclass of*ActivityInstrumentationTestCase2*. The following figures show the procedure to create the *SampleTestUnit* class.







As shown in the previous figure, replace the generic class 'T' with the *SampleJUnitActivity* class. Before implementing the *SampleTestUnit* class, review the methods and its responsibilities.
SampleTestUnit: Defines the constructor for the *SampleTestUnit* class. Android testing framework requires the constructor to initialize the class.
setUp(): This method overrides the JUnit *setUp()* method. Developers can use this method to initialize the environment before each testcase run. Setup method gets called before execution of each test method.

By convention, the test method names are prefixed with "test". These conventions help JUnit to discover the test methods.

- *testViews ():* A testcase that ensures the *SampleJUnit* application starts up correctly with the available UI views.
- *testKilosToPounds():* This tests the conversion of the entered kilos to pounds. As a testcase, give some input in the kilos edittext and check the corresponding result.
- *testPoundsToKilos():* This tests the conversion of the given/entered pounds to kilos. As a testcase, give some input in the pounds edittext and check the corresponding result.
-

As discussed here, implement all the four methods in your testcase class.

Testcase Constructor

The Android testing framework uses a testcase class constructor to run the test. The constructor should call a super constructor with parameters as package name and the class name that tells the framework what needs to be tested.

```
public SampleTestUnit() {
         super("com.sample.junit", SampleJUnitActivity.class);
         // TODO Auto-generated constructor stub
```

```
}
```

setup method

Developers can use the *setup()* method to initialize the variables and test environment.

```
private SampleJUnitActivity mActivity;
private TextView mTextView;
private Button mCKilos,mCPounds;
private EditText mKilos,mPounds;

@Override
protected void setUp() throws Exception {
// TODO Auto-generated method stub
super.setUp();
mActivity = this.getActivity();
mTextView = (TextView) mActivity.findViewById(com.sample.junit.R.id.resultview);
mCKilos = (Button) mActivity.findViewById(com.sample.junit.R.id.Kilos);
mCPounds = (Button) mActivity.findViewById(com.sample.junit.R.id.Pounds);
mKilos = (EditText) mActivity.findViewById(com.sample.junit.R.id.inputvalueK);
mPounds = (EditText) mActivity.findViewById(com.sample.junit.R.id.inputvalueP);
}
```

teardown method

Developers can use the *teardown()* method to make sure that all resources are cleaned up and garbage collected before moving on to the next test.

```
@Override
protected void tearDown() throws Exception {
        // TODO Auto-generated method stub
        //close all resources over here
        super.tearDown();
}
```

testViews case

A testViews method ensures the initial application conditions before executing other tests. This test method is simpler and ensures that the target Views exist before executing the other test methods. One can use Annotations such as SmallTest, LargeTest, MediumTest present inhttp://developer.android.com/reference/java/lang/annotation/Annotation.html

```
@SmallTest
public void testViews() {
assertNotNull(getActivity());
assertNotNull(mTextView);
assertNotNull(mCKilos);
assertNotNull(mCPounds);
assertNotNull(mKilos);
assertNotNull(mPounds);
}
```

testKilosToPounds case

Now add a simple unit test to the test case class. The method *testKilosToPounds()* calls a

JUnit *Assert* method to check whether the target result is according to the expected result.

```
@SmallTest
public void testKilosToPounds() {
    // clearing the mKilos edit text
    mKilos.clearComposingText();
    // tapping the mKilos edit text through TouchUtils class
    TouchUtils.tapView(this, mKilos);
    // sending input to mKilos as 1
    sendKeys("1");
    // Clikcing on the button
    TouchUtils.clickView(this, mCPounds);
    double pounds;
    try {
    // getting the input from the mTextView reference
      pounds = Double.parseDouble(mTextView.getText().toString());
    } catch (NumberFormatException e) {
      pounds = -1;
    }
    // checking the pounds value with the target value.
    assertTrue("1 kilo is 2.20462262 pounds", pounds > 2.2 && pounds < 2.3);
}
```

testPoundsToKilos case

Add one more unit test to the testcase class. The method *PoundsToKilos()* calls a JUnit *Assert* method to check whether the target result is according to the expected result.

```
@SmallTest
public void testPoundsToKilos() {
  // clearing the mPounds edit text
  mPounds.clearComposingText();
  // tapping the mPounds edit text through TouchUtils class
  TouchUtils.tapView(this, mPounds);
  // sending input to mPounds as 1
  sendKeys("1");
  // Clikcing on the button
  TouchUtils.clickView(this, mCKilos);
  double kilos;
  try {
  // getting the input from the mTextView reference
        kilos = Double.parseDouble(mTextView.getText().toString());
  } catch (NumberFormatException e) {
        kilos = -1;
  }
  // checking the kilos value with the target value.
  assertTrue("1 pound is 0.45359237 kilos", kilos > 0.4 && kilos < 0.5);
```

```
}
```

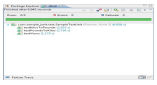
The following code snippet shows the final testcase class:

```
public class SampleTestUnit extends

ActivityInstrumentationTestCase2<samplejunitactivity></samplejunitactivity> {
private SampleJUnitActivity mActivity;
private TextView mTextView;
private Button mCKilos,mCPounds;
private EditText mKilos,mPounds;
public SampleTestUnit() {
         super("com.sample.junit", SampleJUnitActivity.class);
         // TODO Auto-generated constructor stub
}
@Override
protected void setUp() throws Exception {
// TODO Auto-generated method stub
super.setUp();
mActivity = this.getActivity();
mTextView = (TextView)
mActivity.findViewById(com.sample.junit.R.id.resultview);
mCKilos = (Button)
                 mActivity.findViewById(com.sample.junit.R.id.Kilos);
mCPounds = (Button)
                 mActivity.findViewById(com.sample.junit.R.id.Pounds);
mKilos = (EditText)
                 mActivity.findViewById(com.sample.junit.R.id.inputvalueK);
mPounds = (EditText)
                 mActivity.findViewById(com.sample.junit.R.id.inputvalueP);
}
@SmallTest
public void testViews() {
assertNotNull(getActivity());
assertNotNull(mTextView);
assertNotNull(mCKilos);
assertNotNull(mCPounds);
assertNotNull(mKilos);
assertNotNull(mPounds);
}

@SmallTest
public void testKilosToPounds() {
mKilos.clearComposingText();
```

```
TouchUtils.tapView(this, mKilos);
sendKeys("1");
TouchUtils.clickView(this, mCPounds);
double pounds;
try {
           pounds = Double.parseDouble(mTextView.getText().toString());
} catch (NumberFormatException e) {
pounds = -1;
}
assertTrue("1 kilo is 2.20462262 pounds", pounds > 2.2 && pounds < 2.3);
}
@SmallTest
public void testPoundsToKilos() {
mPounds.clearComposingText();
TouchUtils.tapView(this, mPounds);
sendKeys("1");
TouchUtils.clickView(this, mCKilos);
double kilos;
try {
kilos = Double.parseDouble(mTextView.getText().toString());
} catch (NumberFormatException e) {
kilos = -1;
}
assertTrue("1 pound is 0.45359237 kilos", kilos > 0.4 && kilos < 0.5);
}
}
```

Running the Test Project

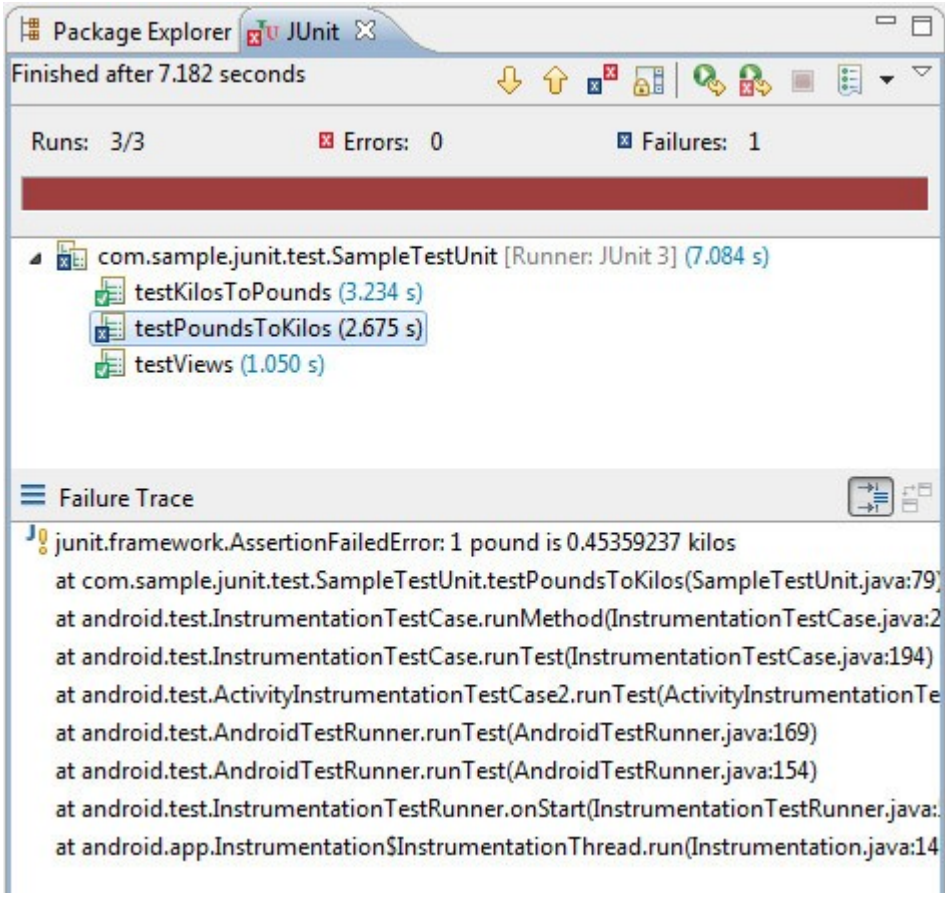
Run your test project to check the test cases. For this, right click on **SampleJUnitTest** and select **Run As -> Android JUnit Test**. The following screen is the result if your application compiles properly:



This figure shows no errors. Tweak the code to generate errors so you can see the difference. Change the code in*testPoundsToKilos()* method

```
public void testPoundsToKilos () {
...
...
...
assertTrue("1 pound is 0.45359237 kilos", kilos > 0.5 && kilos < 0.6);
...
}
```

If you run the testcases now, you get the following screen (or something similar):



ref:http://developer.samsung.com/android/technical-docs/Basics-of-JUnit-in-Android