# SEGR 5210 Software Testing
# Fall 2014

# Robotium Tutorial

This tutorial builds on the Android unit testing tutorial with the NumberGuessing app. You must have completed that tutorial before starting this one.

## Robotium

According to its website:

> Robotium is an Android test automation framework that has full support for native and hybrid applications. Robotium makes it easy to write powerful and robust automatic black-box UI tests for Android applications. With the support of Robotium, test case developers can write function, system and user acceptance test scenarios, spanning multiple Android activities.

Robotium has two distinct advantages over the usual Android testing framework that is part of Eclipse:

- Robotium provides a set of functions that make testing complex scenarios more straightforward, especially for tests that span multiple activities.
- Robotium allows you to test Android programs without having the source code.

This tutorial focuses on the first advantage. For more information on Robotium, check out the website: https://code.google.com/p/robotium/

## Downloading and Setting Up The Project

To start, download the Robotium jar file from the website https://code.google.com/p/robotium/ and place it in a directory. These steps are needed to set up the test project to use Robotium:

1. In Package Explorer, right-click on the NumberGuessingTest project and select Properties.
2. Select Java Build Path → Libraries tab → Add External JARs…
3. Browse to the directory when you placed the Robotium jar file and select the .jar file. It should appear in the list of libraries.
4. Select the Order and Export tab and check the box next to the Robotium jar file.
5. Press Ok.

## Creating the Test Class

Robotium tests use the `ActivityInstrumentationTestCase2` class as Android unit tests. The process of creating a test class is the same. To review:

1. In Package Explorer, right-click on the NumberGuessingTest project and select New → Class.

2. In the dialog box:
    a. Package: `edu.seattleu.numberguessing.test`
    b. Name: `RobotiumTest`
    c. Superclass: `android.test.ActivityInstrumentationTestCase2<MainActivity>`
    d. Click Finish.
3. This will create a blank class with two errors.
    a. No import of the subject under test (`edu.seattleu.numberguessing.MainActivity` in this case). This error can be fixed by hovering over the red squiggly line and picking the first option in the quick fix menu.
    b. No constructor. Add the following constructor:

```
public RobotiumTest() {
   super(MainActivity.class);
}
```

***Set Up***

The next step is to create a `setUp` function.

```
@Override
protected void setUp() throws Exception {
      super.setUp();
      solo = new Solo(getInstrumentation(),getActivity());
      mActivity = getActivity();

      mActivity.setMysteryNumber("5913");

      mPrevGuess = (TextView) solo.getView(R.id.prevGuess);
      mResult = (TextView) solo.getView(R.id.result);
      mGuessCount = (TextView) solo.getView(R.id.guessCount);
      mGuessEntry = (EditText) solo.getView(R.id.guessEntry);
}

private Solo solo;
private MainActivity mActivity;
private TextView mPrevGuess;
private TextView mResult;
private TextView mGuessCount;
private EditText mGuessEntry;
```

This is similar to the setup for Android tests:

- The first three lines will be the same for all tests.
- The Solo class is used to direct UI flow within a test.
- As before, the mystery number is initialized in this routine.
- Unlike before, it is not necessary to create members for the buttons.
- Another change from the previous tutorial is that I'm not using fully qualified names for the GUI elements. I'm using shorter names by adding this import (I should have done in the previous tutorial):

```
import edu.seattleu.numberguessing.R;
```

For Robotium tests, you will need this `tearDown` function:

```
@Override
protected void tearDown() throws Exception {
	solo.finishOpenedActivities();
}
```

Here are the same tests from the previous tutorial written using Robotium:

```
private static final int TIMEOUT_IN_MS = 5000;

public void testRestart() {
	solo.clickOnButton("Submit");
	solo.clickOnButton("Restart");
	checkGuess("", "", "0");
}

public void testGuessCount() {
	for (int i = 0; i < 4; i++)
		solo.clickOnButton("Submit");
	checkGuess("0000", "18", "4");
}

public void testSubmit() {
	makeGuess("1234");
	checkGuess("1234", "14", "2");
}

public void testWinGame() {
	makeGuess("5913");
	solo.assertCurrentActivity("expected WinningMsgActivity",
		WinningMsgActivity.class);
}

private void makeGuess(final String guess) {
	solo.enterText(mGuessEntry, guess);
	solo.clickOnButton("Submit");
}

private void checkGuess(String prevGuess, String result, String guessCount)
{
	solo.sleep(TIMEOUT_IN_MS);
	assertEquals(prevGuess, mPrevGuess.getText().toString());
	assertEquals(result, mResult.getText().toString());
	assertEquals(guessCount, mGuessCount.getText().toString());
}
```

Some nice things associated with Robotium:

- To select buttons, the parameter is the text on the button. This feature allows you to select buttons even when you do not have the source code.
- Entering tests is much easier. Simply use the `enterText` method.
- Detecting a change in the screen is simple using the `assertCurrentActivity` method.

Other notes:

- The `testRestart` method uses the `checkTest` helper method. I could have done that in the previous tutorial.
- The `solo.sleep` method is needed because the `assertEquals` execute before the GUI has a had a chance to update. This is not the recommended method – it is better to use one of the wait functions. However, different situations require different wait methods. The sleep method is used here because it should work in nearly all situations.
- The `testSubmit` method checks that the number of guesses is equal to two instead of one. There are two reasons why: (a) the `enterText` method first registers a mouse button press input in the text area before supplying text and (b) the number guessing program registers the click as a guess (it probably should not do that in hind sight but I'm calling it a "feature" instead of a bug right now). As a result, two guesses are made.

The beauty of Robotium is to create tests that span multiple screens. This is very difficult to do with the default Android testing framework. Here is a test where the winning number is entered taking the user to the winning message screen and then the user presses the "Play Again" button which will start a new game.

```
public void testWinGameScenario() {
      makeGuess("5913");

      solo.assertCurrentActivity("expected WinningMsgActivity",
            WinningMsgActivity.class);
      TextView guessText = (TextView) solo.getView(R.id.guessText);
      assertEquals("2", guessText.getText().toString());
      solo.clickOnButton("Play Again");

      solo.assertCurrentActivity("expected MainActivity",
            MainActivity.class);
      checkGuess("", "", "0");
}
```