Robotium Tutorial: Your First Android Framework

What is the Robotium?

Robotium is an android Testing framework to automate test cases for native and hybrid applications. Using Robotium, the developer can create strong automatic GUI testing case for Android applications. In addition, the developer could write a functional, system and acceptance test scenario, spreading many Android activities.

In this tutorial, you will learn-

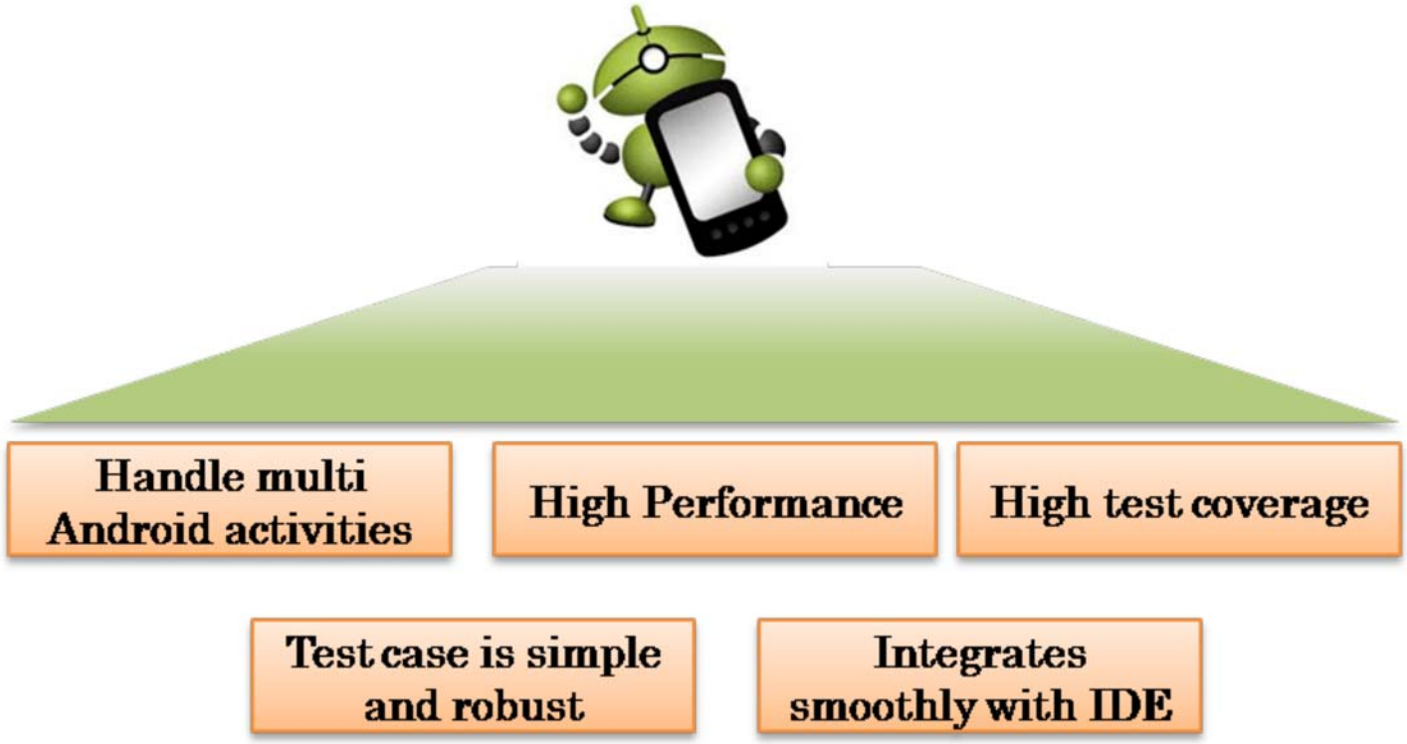
Robotium testing framework

Standard Android testing framework has some limitation as below

- Unable to handle multiple activities
- Test execution performance is slow
- Test cases are complex & hard to implement

Robotiumframework is the better choice to conduct testing on Android application

Robotium is open source framework and is considered an extension of Android test framework. Using Robotium, developer can create robust automatic GUI test cases for Android applications. Moreover, developer can write functional, system and acceptance test scenarios, spanning multiple Android activities.

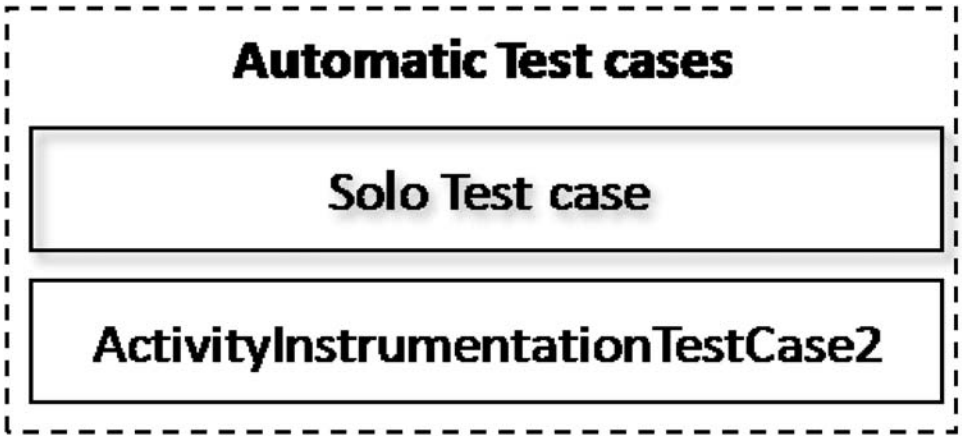


(/images/image017(4).png)

Advance features of Robotium

Robotium Test Case Classes

Robotium uses set of classes (**com.jayway.android.robotium.solo**) for testing**.** This class supports test cases that span over multiple activities. Solo is integrated with the ActivityInstrumentationTestCase2.

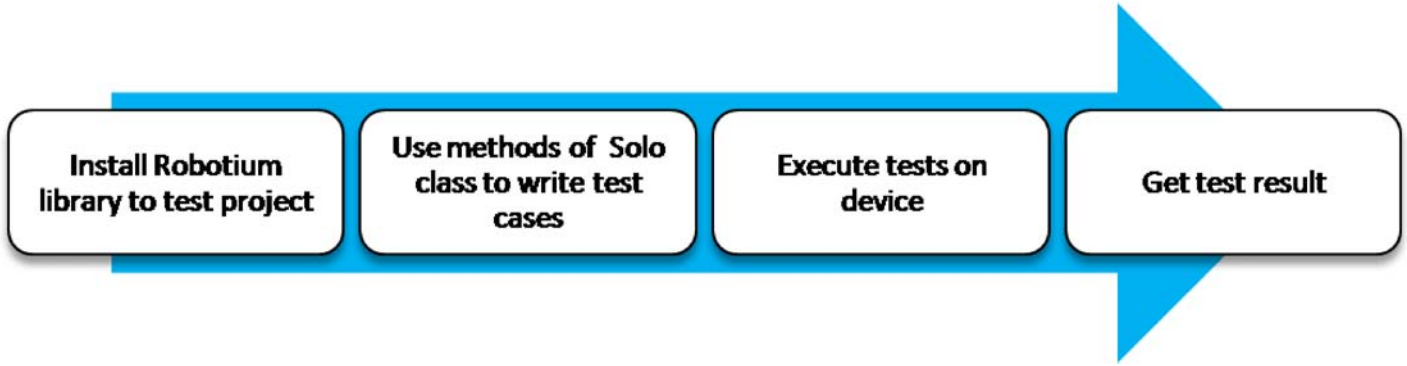


(/images/image019(3).png)

Tester can write test cases without knowledge of application design (black box testing) by using Robotium test case classes. It is an outstanding feature compare to Android test case classes.

How to use Robotium

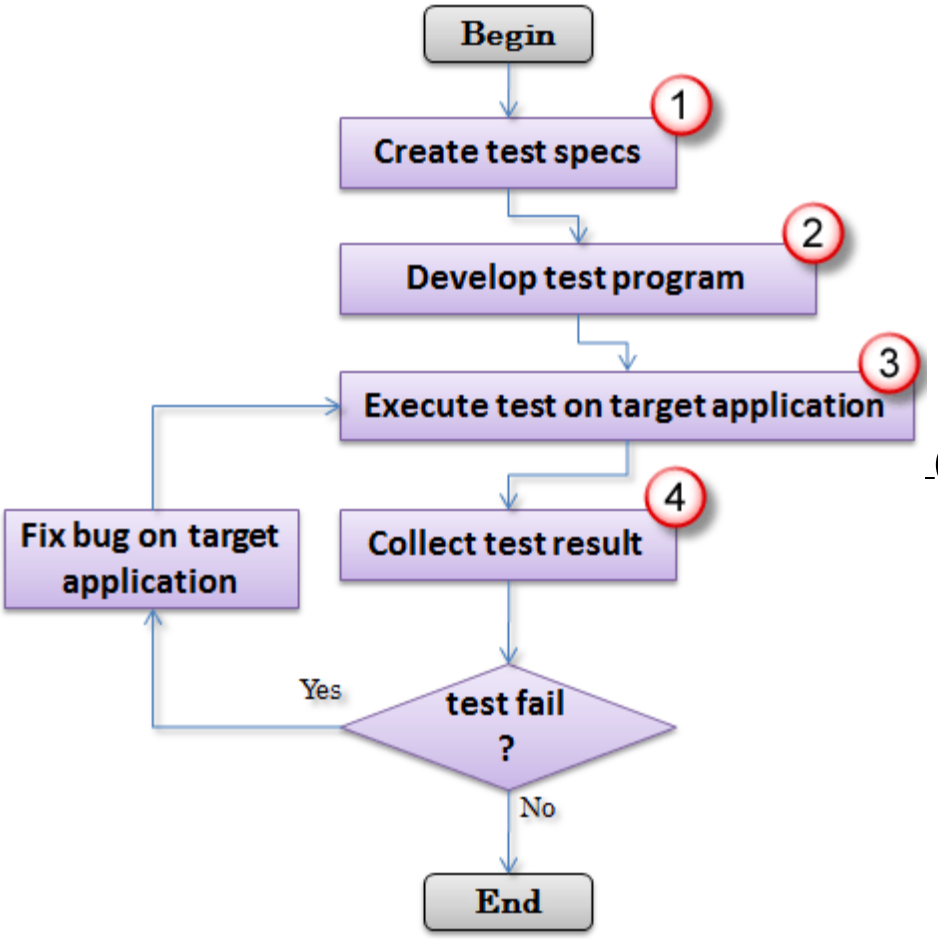
To use Robotium in your Android test project, you need follow the steps below



(/images/image021(3).png)

Using Robotium to conduct testing on Android application. To guarantee quality of your Android application, you should follow the procedure below

1. Design test specification
2. Develop test program
3. Execute Test Case (/test-case.html)on target device
4. Collect test result



(/images/flow_chart.png)

Android application Testing (/software-testing.html)procedure

STEP 1) Design test specification

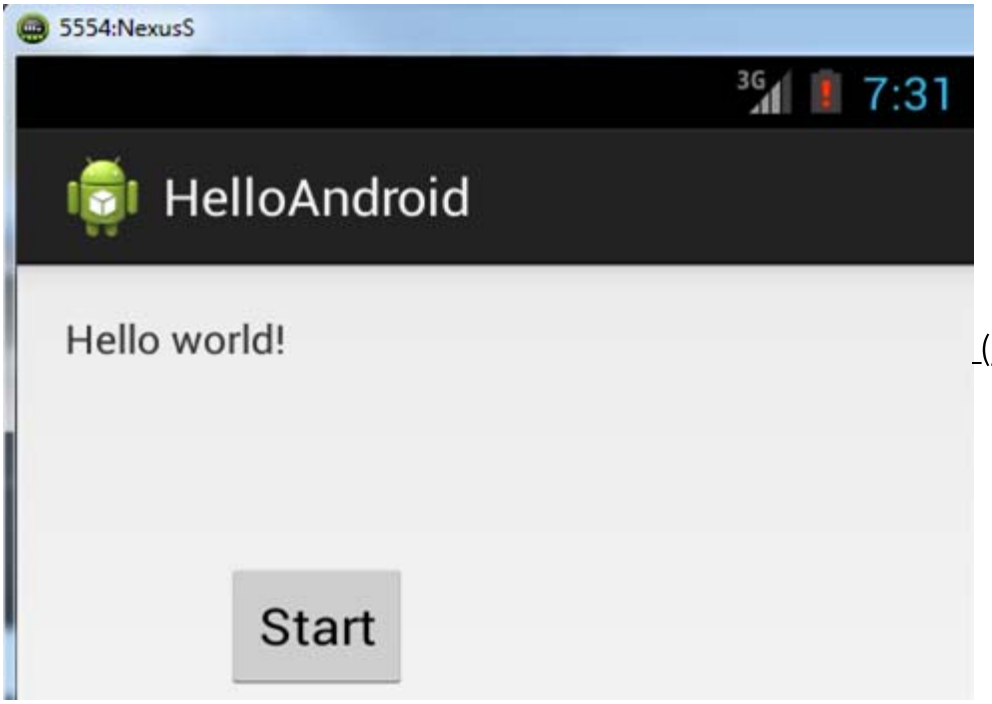
- This is the first step to test your application.In this step you Define target to be test. In your Android application, there's many targets need to be tested such as UI, Activity, components, services. Clearly defining the target in your application will help achieve wide test coverage.
- Plan the test types should be conducted (Unit test, Functional test, System test).
- Design test cases for maximum coverage but minimize number of test cases. The more code is tested more are chances of early bug detection.

STEP 2) Write TEST program

This section guides you how to write an Android test program using Android Junit (/junit-tutorial.html)Test and Robotium. Assume that you have already developed an Android program name *HelloAndroid.* This program has some functions described below:

- Display a text "Hello world!" on screen.

- Display a message HelloAndroid when user press "Start" button



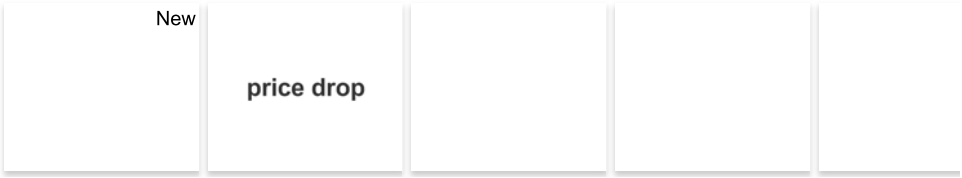
(/images/hello_world.png)

HelloAndroid Application

System Requirements

- Android platform comes with pre-integrated JUnit 3.0 framework.
- In order to create Android Test Project from Eclipse, your computer must have installed:
  - Latest version Android Platform (currently Android 8.1)

You can download (http://developer.android.com/sdk/index.html) Eclipse IDE with built-in ADT (Android Developer Tools). It includes the essential Android SDK components and a version of the Eclipse IDE .

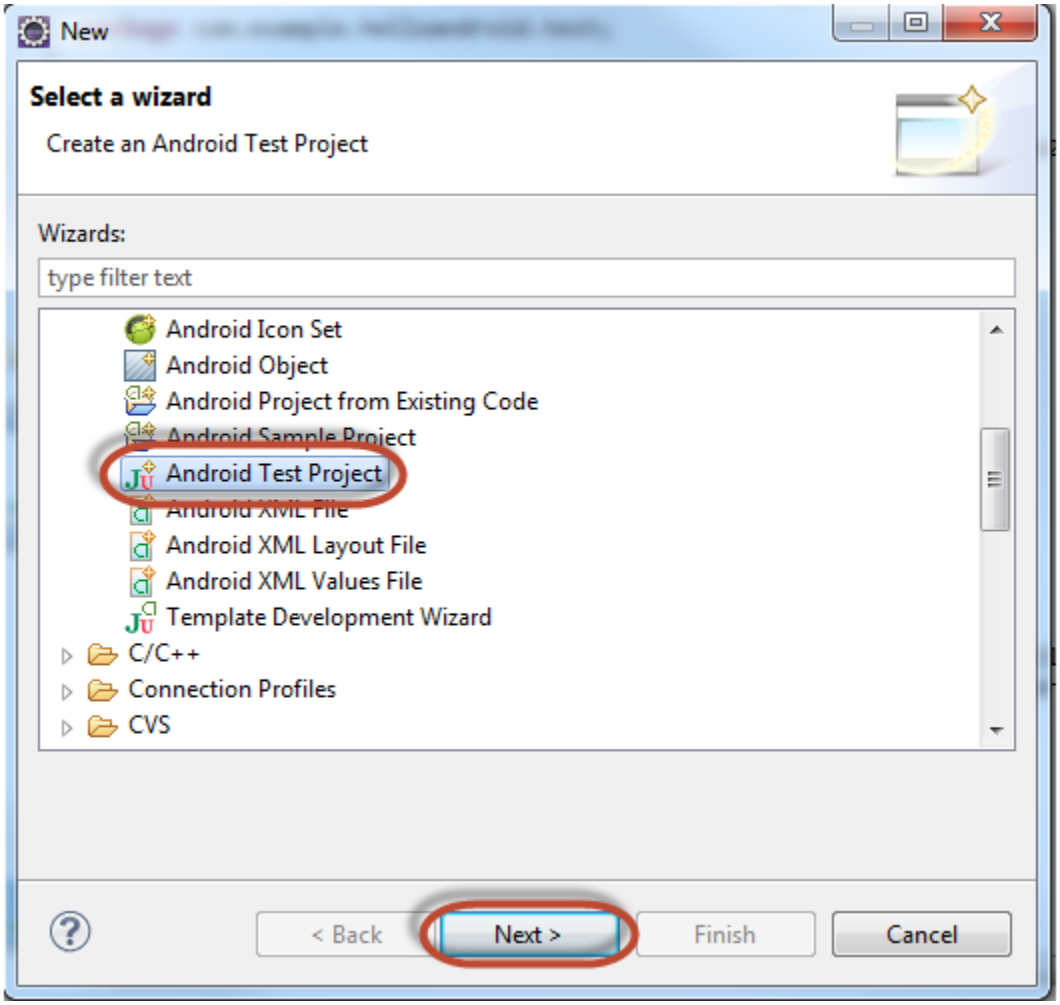


For the Robotium testing framework, you need to down Robotium library from Robotium webpage (https://github.com/robotiumtech/robotium).

Create Android Test Project

- Click File -> New -> Other

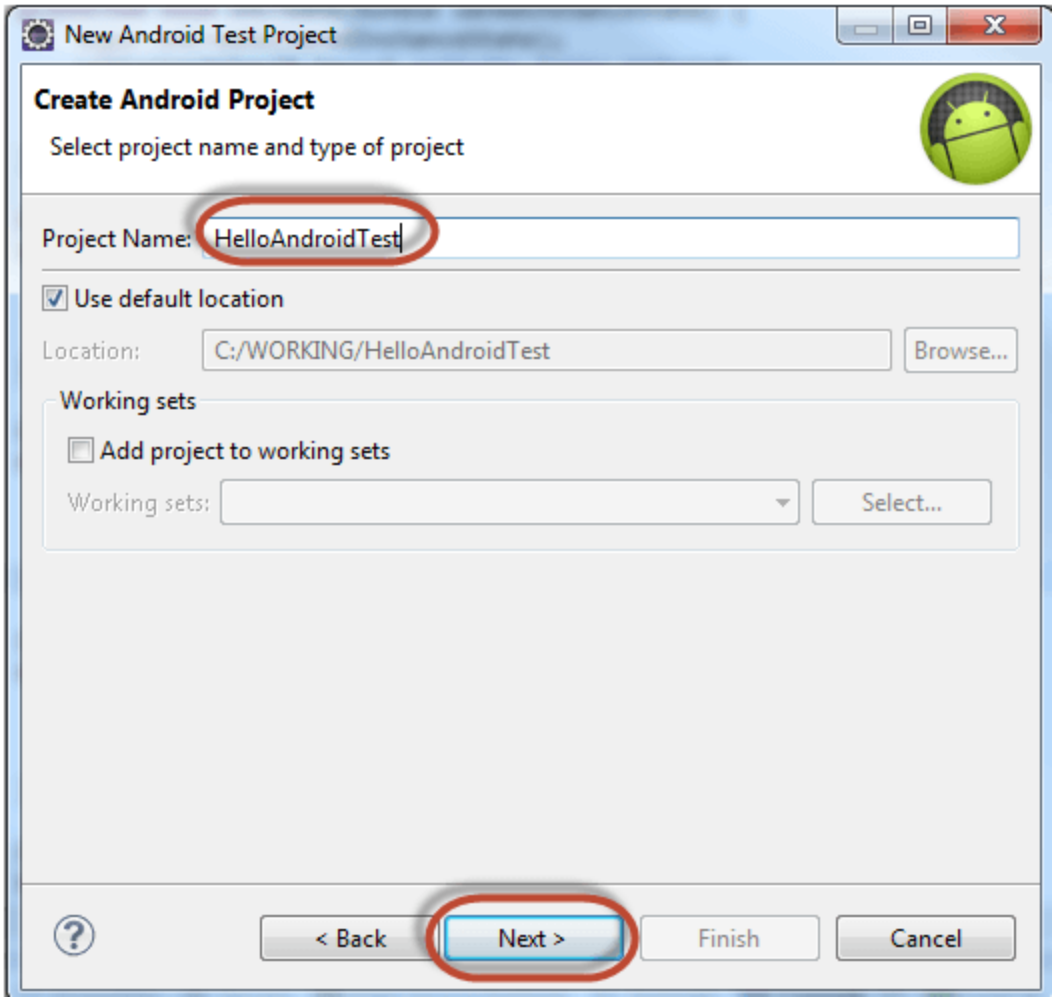
- Choose: Android -> Android Test Project as per below figure -> Choose Next



(/images/select_wizard.png)

Create new Android test project

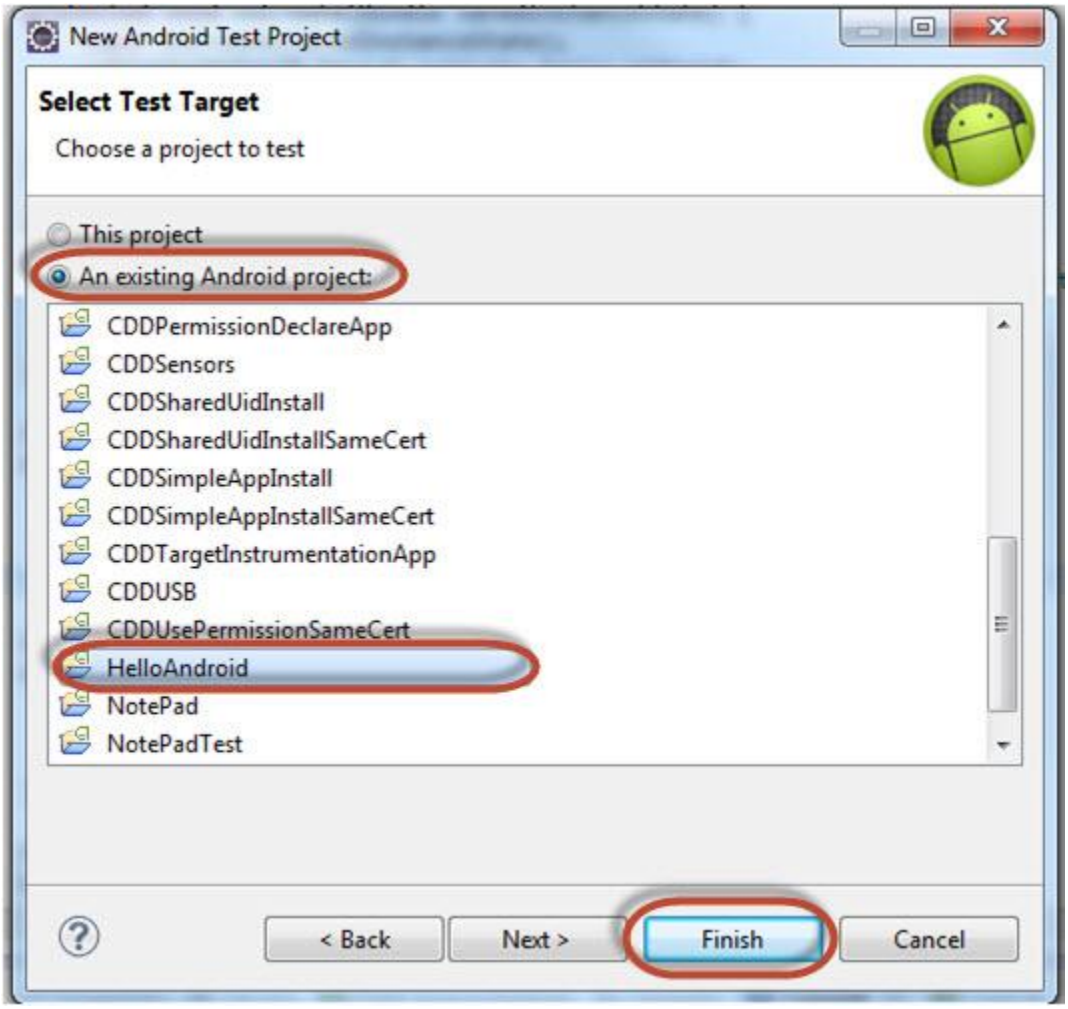
Write name of your test project. As naming convention, your test project should be name "*HelloAndroidTest*"



(/images/crearte_android(1).png)

Add test project name base on naming convention

Choose target application under test. In this case, this is HelloAndroid click Finish



(/images/test_targetjpg.jpg)

Choose target application under test

Create Test Suites

Base on your test specification, you started to create test suites for your test program. You can choose various Testing framework. In this tutorial, I choose standard Android testing framework **ActivityInstrumentationTestCase2**. You have to add Robotium library file to a *libs* directory in your project folder in case you want to test with Robotium framework. (You create lib folder in your project folder).

A test case defines the fixture to run multiple tests. To define a test case, you must follow the program structure below:

- Implement a subclass of `TestCase`.
- Define instance variables that store the state of the fixture
- Initialize the fixture state by overriding setUp() (http://developer.android.com/reference/junit/framework/TestCase.html#setUp%28%29)

- Clean-up after a test by overriding tearDown() (http://developer.android.com/reference/junit/framework/TestCase.html#tearDown%28%29).

```
7
8  public class HelloAndroidTest extends ActivityInstrumentationTestCase2<HelloAndroid> {
9
10     private HelloAndroid mActivity;
11     private TextView mView;
12     private String resourceString;
13
14     public HelloAndroidTest () {
19      @Override
20     protected void setUp() throws Exception {
21         // TODO Auto-generated method stub
22         super.setUp();
23         try {
24             mActivity = this.getActivity();
25             mView = (TextView) mActivity.findViewById(com.example.helloandroid.R.id.textvie
26             resourceString = mActivity.getString(com.example.helloandroid.R.string.hello_wo
27
28         } catch (Exception e) {
29             e.printStackTrace();
30         }
31     }
32
33      @Override
34     protected void tearDown() throws Exception {
35         // TODO Auto-generated method stub
36         super.tearDown();
37     }
```

Activity need to test

(/images/image009(6).png)

Test program's structure

```
package com.example.helloandroid.test;

import com.example.helloandroid.HelloAndroid;
import com.jayway.android.robotium.solo.Solo;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class HelloAndroidTest extends ActivityInstrumentationTestCase2 <HelloAndroid> {

        private HelloAndroid mActivity;
        private TextView mView;
        private String resourceString;
        private Solo solo;

        public HelloAndroidTest () {
                // TODO Auto-generated constructor stub
                super("com.example.helloandroid",HelloAndroid.class);
        }

         @Override
        protected void setUp() throws Exception {
                // TODO Auto-generated method stub
        //      super.setUp();

                mActivity = this.getActivity();
                solo = new Solo(getInstrumentation(),getActivity());
                mView = (TextView) mActivity.findViewById(com.example.helloandroid.R.id.textvie
w2);
                resourceString = mActivity.getString(com.example.helloandroid.R.string.hello_wo
rld);

        }

         @Override
        protected void tearDown() throws Exception {
                // TODO Auto-generated method stub
                //super.tearDown();
                solo.finishOpenedActivities();
        }

        public void testPrecondition() {
                assertNotNull(mView);
        }

        /* test Target application contains a text display "Hello World!"*/
        public void testSearchText() {
                assertEquals(resourceString,(String) mView.getText());
        }

        /* test HelloAndroid Activity on target application is exist*/
```

```
        public void testCurrentActivity() throws Exception  {
        solo.assertCurrentActivity("wrong activity", HelloAndroid.class);
    }

        /* test Application UI contains "Start" button */
        /* send event click button to target application */
    public void testSearchButton() throws Exception {
        boolean found = solo.searchButton("Start");
        solo.clickOnButton("Start");
        assertTrue(found);
    }



}
```

Adding Test Cases

- In the same package with TestSuite, we create TestCase classes
- To test certain activity i.e. *HelloAndroid*, create a test case extent *ActivityInstrumentationTestCase2<HelloAndroid>*
- In this class, tester can obtain testing activity through getActivity() method .
- You can freely create test for a testing activity by create method with name "test + original Method Name"
- In test method, tester can use Android JUnit function to compare the actual value and expected value. These methods are shown in below.

```
/* test Target application contains a text display "Hello World!"*/
public void testSearchText() {
    assertEquals(resourceString,(String) mView.getText());
}

/* test HelloAndroid Activity on target application is exist*/
public void testCurrentActivity() throws Exception  {
    solo.assertCurrentActivity("wrong activity", HelloAndroid.class);
}

/* test Application UI contains "Start" button */
/* send event click button to target application */
public void testSearchButton() throws Exception {
    boolean found = solo.searchButton("Start");
    solo.clickOnButton("Start");
    assertTrue(found);
}
```

(/images/image010(5).png)

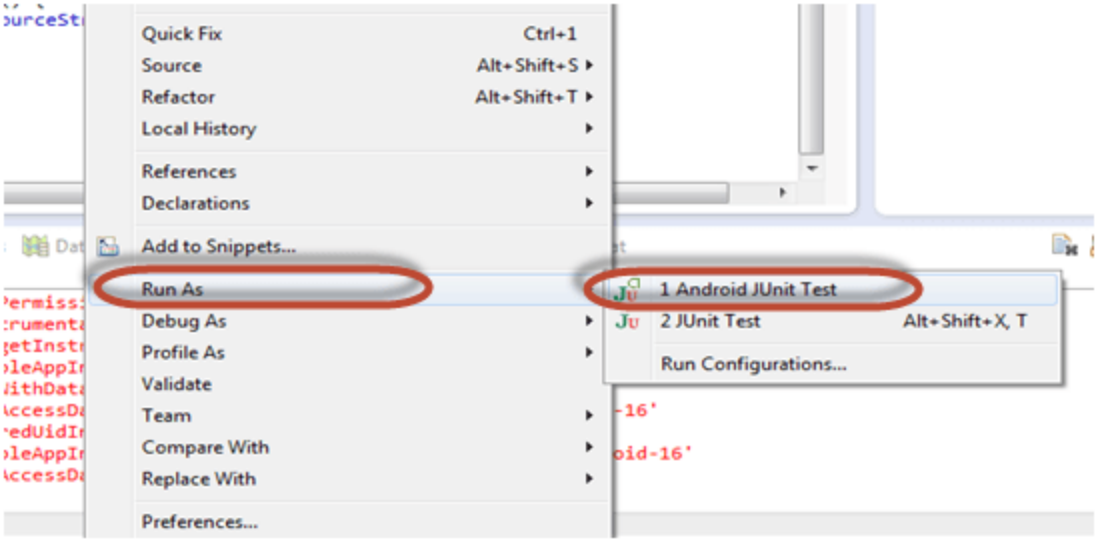
Example methods of Robotium and Android Testing framework

These test suites above verified that Application GUI must display a text "*Hello World!*", and contains a button name "*Start*".

STEP 3) Run Test

After you finish writing your test program, run the test using the steps below

- Connect Android device to your PC (or start Emulator in case you don't have real device).
- In your IDE, right click àRun asàAndroid Unit Test



(/images/Run_test.png)

Running test program

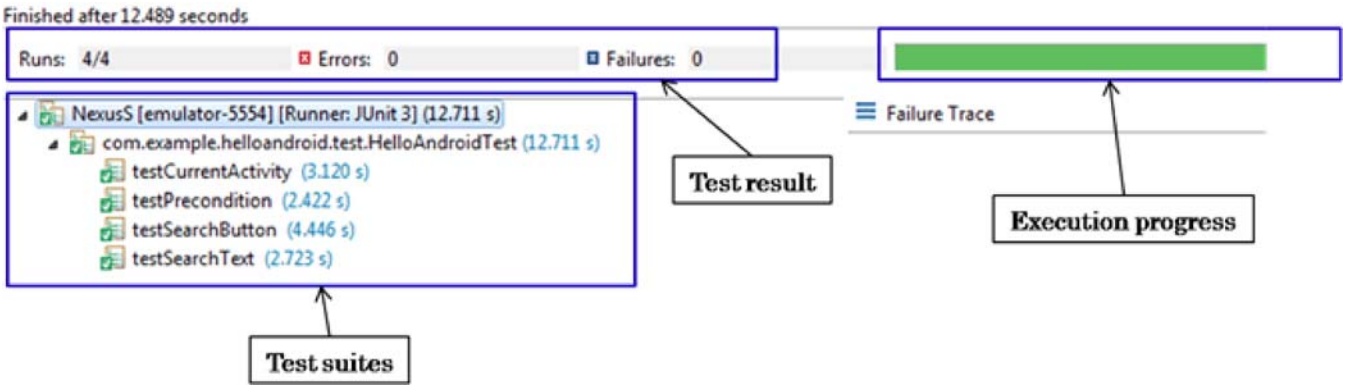
Besides running test on IDE, you can run test on command line. In this test program, test package is *com.example.helloandroid.test* . In **Linux** terminal, you can use following command to run all test in this package:

$ adb shell am instrument -w -e package com.example.helloandroid.test

STEP 4) Get test result

After test executes, you get test results .

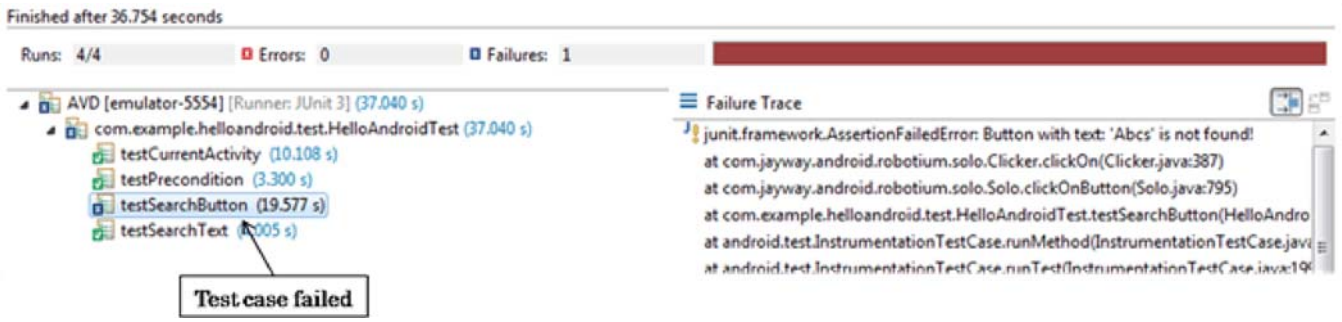
In this test program, 4 test methods are executed. In this case, all test cases are passed.



(/images/image014(4).png)

Test result output in case all test cases passed

In case test case fails, the output is display and show you which test cases failed



(/images/image016(4).png)

Test result output in case all test cases failed

Source code examples

This articles include some Source Code examples which help you to understand the tutorial more clearly and quickly catch up the technical knowledge

- HelloAndroid (https://drive.google.com/uc?export=download&id=0B_vqvT0ovzHca3Fpb3RjN2Jfb2c): Application under test .
- HelloAndroidTest (https://drive.google.com/uc?export=download&id=0B_vqvT0ovzHcYW5LTWdhQ2pseTg): Test program using Android Test framework

YOU MIGHT LIKE:

MOBILE TESTING

(/mobile-app-performance-testing-  strategy-tools.html) (/mobile-app-performance-testing-strategy-tools.html)

Mobile App Performance Testing: CheckList, Tools (Andriod & iOS)

(/mobile-app-performance-testing-strategy-tools.html)

MOBILE TESTING

(/appium-maven.html)  (/appium-maven.html)

Appium Maven Dependency

(/appium-maven.html)

MOBILE TESTING

(/game-testing-mobile-  desktop-apps.html) (/game-testing-mobile-desktop-apps.html)

Game Testing: How to Test Mobile/Desktop Apps

(/game-testing-mobile-desktop-apps.html)

MOBILE TESTING

 (/adb-connect.html) (/adb-connect.html)

Connect Mobile using Android Debug Bridge(ADB) over USB & WiFi

(/adb-connect.html)

MOBILE TESTING

(/mobile-testing-interview-  questions.html) (/mobile-testing-interview-questions.html)

Top 20 Mobile Testing Interview Questions & Answers

(/mobile-testing-interview-questions.html)

MOBILE TESTING

(/interrupt-testing.html)  (/interrupt-testing.html)

Interrupt Testing in Mobile Application

(/interrupt-testing.html)

Mobile Testing

5) Appium Desired Capabilities (/appium-desired-capabilities.html)

6) Connect Mobile using ADB (/adb-connect.html)

7) Appium Maven Dependency (/appium-maven.html)

8) Appium Interview Q & A (/appium-interview-questions.html)

9) Android Testing & Automation (/why-android-testing.html)

10) Robotium Tutorial (/first-android-testing.html)

11) Introduction to Selendroid (/introduction-to-selendroid.html)

12) Mobile App Performance Testing (/mobile-app-performance-testing-strategy-tools.html)

13) Calabash Tutorial for Android (/calabash-android-ios-testing.html)

14) iOS App Testing (/getting-started-with-ios-testing.html)

15) iOS Automation Testing (/ios-test-program-uiautomation-framework.html)

¶

 (https://www.facebook.com/guru99com/) 
(https://twitter.com/guru99com) 
(https://www.youtube.com/channel/UC19i1XD6k88KqHlET8atqFQ)

(https://forms.aweber.com/form/46/724807646.htm)

About

About Us (/about-us.html)
Advertise with Us (/advertise-us.html)
Write For Us (/become-an-instructor.html)
Contact Us (/contact-us.html)

Career Suggestion

SAP Career Suggestion Tool (/best-sap-module.html)
Software Testing as a Career (/software-testing-career-complete-guide.html)
Certificates (/certificate-it-professional.html)

Interesting

Books to Read! (/books.html)
Blog (/blog/)
Quiz (/tests.html)
eBook (/ebook-pdf.html)

Execute online

Execute Java Online (/try-java-editor.html)
Execute Javascript (/execute-javascript-online.html)
Execute HTML (/execute-html-online.html)
Execute Python (/execute-python-online.html)