Lab 3.1 – Unit Testing an Android App with Android Studio

Overview

In this lab you will create some unit tests for your AndroidLab project. Android uses the jUnit 3 framework for unit testing.

Setup

Lab 02-1 should be completed so that you have a working app to test.

Steps

1) Create the directory to contain unit tests.

1.a) Inside the module you want to test, navigate to the "src" directory.

1.b) Create a new directory called "androidTest". This will be at the same level as the "main" directory.

> **Note** This directory may already exist depending on how you created the module.

1.c) Inside the "androidTest" directory create a directory called "java". This is where the unit test code will be placed.

1.d) Inside "java" create a package with the same value as in your "main/java" directory. Use the package wizard. Create a package named "com.garagesalesapp".

> **Note** Using the same package name is not required but ensures that the test classes have the save "visibility" as the classes under test.

2) Create a unit test. Unit tests will be placed in the java package under the "androidTest" directory.

Unit tests are created as java code.

2.a) In `src/androidTest/java/com.garagesalesapp` create a java class named "SimpleAndroidTest"

Select "New → Java Class". There is no specific wizard for unit tests.

2.b) Extend the `AndroidTestCase` class

```
public class SimpleAndroidTest extends AndroidTestCase {
}
```

2.c) And a method call "testGetAllEvents"

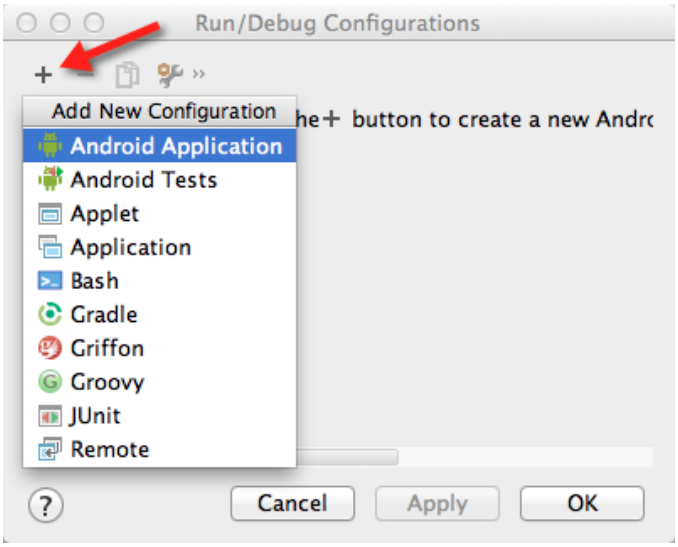
```
public void testGetAllEvents() {
                fail("Not yet implemented");
}
```

This follows the "fail first" strategy of unit testing

3) Run the unit test and verify that it fails. You must create a run configuration to run the test.

3.a) Select "Run → Edit Configurations" from the top level menu.

3.b) Select "+" to add a new run configuration and select "Android Tests"



3.c) Define the test by entering the following values in the wizard:

The wizard should look like this:

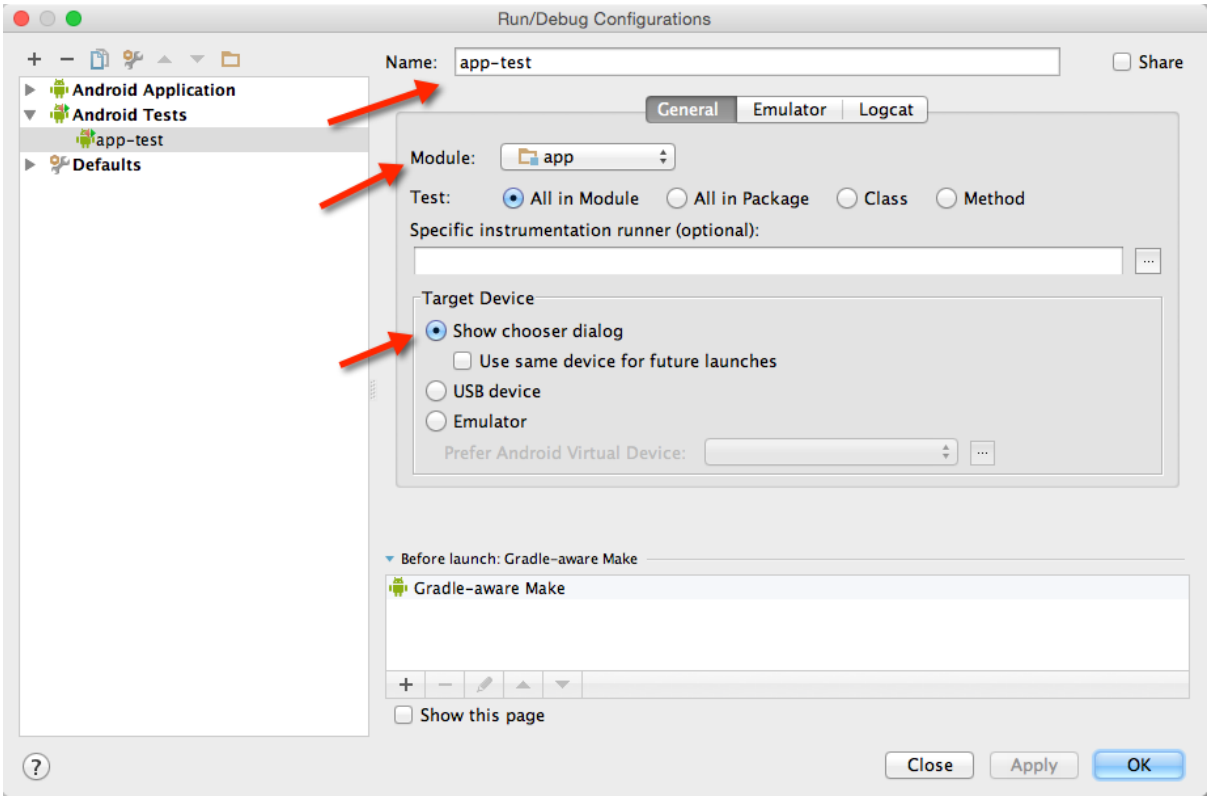
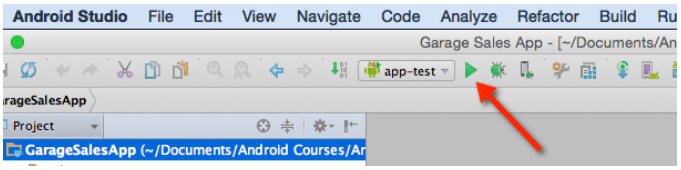


Figure 1. Android Test Wizard

4.d) Run "app-test" using the play button.



Under the "Run" tab at the bottom of the screen you should see the jRunner output:

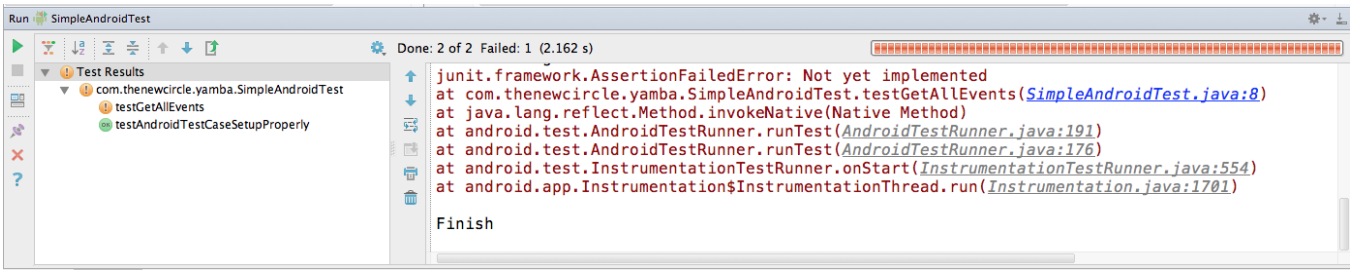


Figure 2. Runner Output

Note: The test failed because you forced it to.

5) Create a working test

5.a) Add the following test method to the test

```
public void testGetAllEvents() {
  Context context = getContext();
  List<Event> events = EventService.getAllEvents(context);
  assertTrue(events.size() > 0);
  assertEquals(15, events.size());
}
```

5.b) Run "app-test" again using the play button.

Under the "Run" tab at the bottom of the screen you should see the jRunner output:

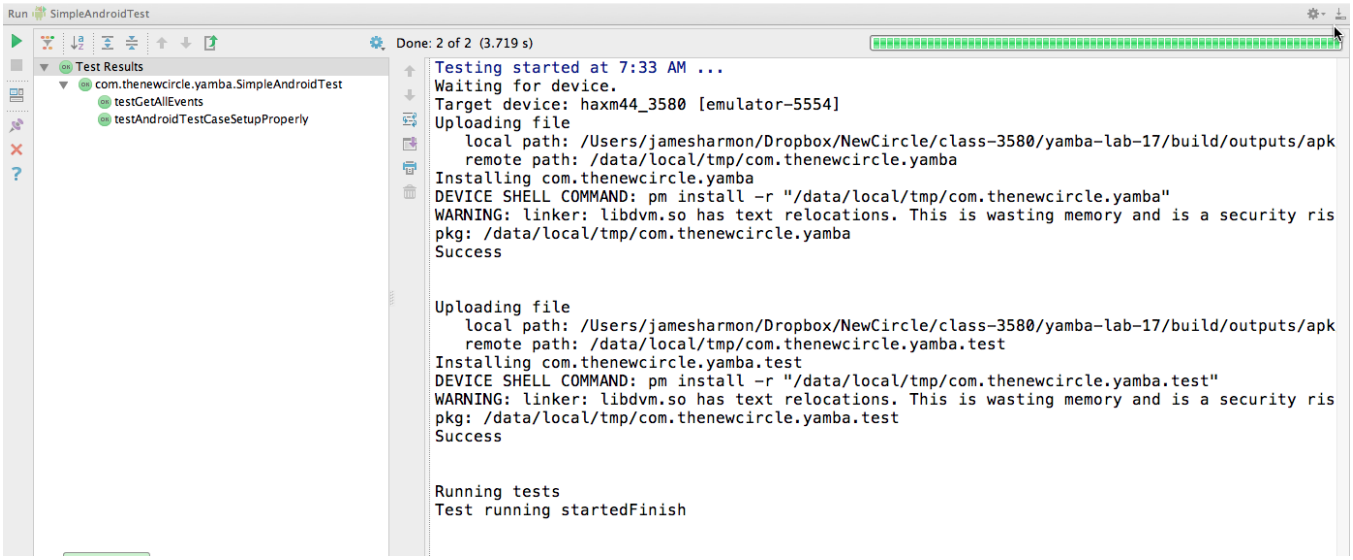


Figure 3. Runner Output

Note: This time you should see a green bar to signify that the test worked (Green is clean!!!, Read id dead!!!)

6.) Create a unit test for activity UI.

Although you've tested the event service directly, as you should also verify that the UI view element that the user sees contains the correct value. Create a unit test to run the actual activity and inspect the UI element.

6.a) Create a new jUnit Test Case for the MainActivity called "MainActivityTest". Extend the activity testing class from `android.test.ActivityInstrumentationTestCase2<MainActivity>`

6.b) Create a new constructor

```
public MainActivityTest() {
   super("com.garagesalesapp", MainActivity.class);
}
```

6.c) Create setup method and an instance variable for the activity.

```
MainActivity mainActivity;

@Override
protected void setUp() throws Exception {
   super.setUp();
   mainActivity = getActivity();
}
```

6.d) Create a new test method called "testActivityUI" which will lookup the text element and check it's value.

```
public void testActivityUI() {
   TextView textView = (TextView) mainActivity.findViewById(com.garagze.R.id.textView)
   assertEquals("Number of events: 15", textView.getText().toString());
}
```

6.e) Create a new run configuration and un the test until you get a green bar.

7.) Test a modification to the UI

The test must run in the main application's thread, also known as the UI thread. The Android testing framework provides an annotation for ensuring that code runs on the main UI thread. Create the following test method to verify that you can change the value of a UI view:

7.a) Create a new method called `testView` and test the value of a view element.

Use the following only as a template - your code will probably be different.

```
@UiThreadTest
public void testView() {
  TextView textView = (TextView) mainActivity.findViewById(R.id.textView);
      textView.setText("abc");
  assertEquals("abc", textView.getText().toString());
}
```

7.b) Run the test until you get a green bar.

LAB 3.1

Last updated 2014-11-18 09:23:15 CST