

Software Engineering Large Practical: Testing Android applications

Stephen Gilmore

School of Informatics

Wednesday 1st November, 2017

1. Software testing
2. Recording user interface tests
3. Running tests

Software testing

Software testing

- In practice, software is produced to tight product deadlines. Changes to product requirements are frequent, often because our software is interacting with other software that is changing.
- The principal method of improving software quality is through **automated testing**, with test frameworks being used to encode tests which can be re-run every time that the code is updated.
- We will only be introducing the very important topic of software testing here, and only specifically for Android applications. For an in-depth look at the subject see the **Software Testing** course next semester.

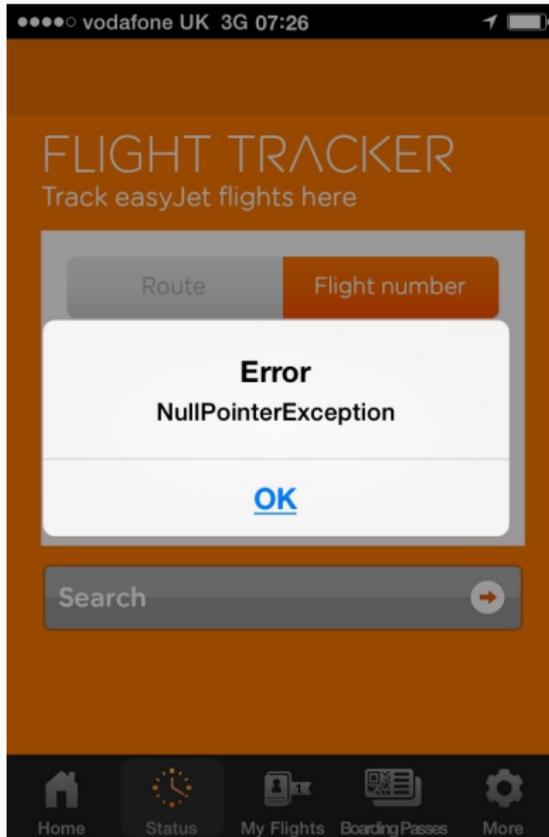
Expectations of software testing

Program testing can be used to show the presence of bugs, but never to show their absence!

— Edsger W. Dijkstra, 1970.

- In 2017, very few people expect to be able to prove that their code has no bugs. **Perfection in software development is not widely viewed as an attainable goal.**
- However, user-visible errors in software cause reputational damage and can cause your app to be uninstalled or receive negative user reviews.
- Pragmatically, the goal of software testing is to **reduce the number of bugs in code which is shipped to the user.**

Needs more testing

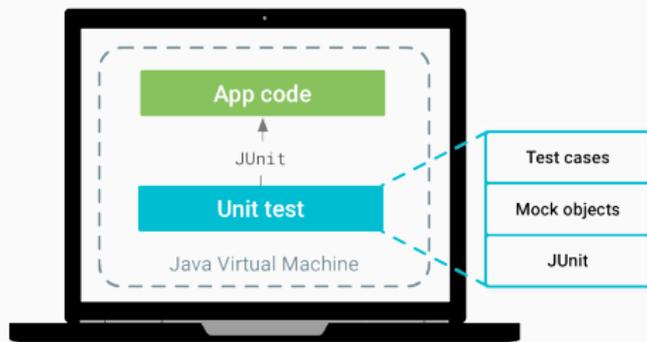


Android testing

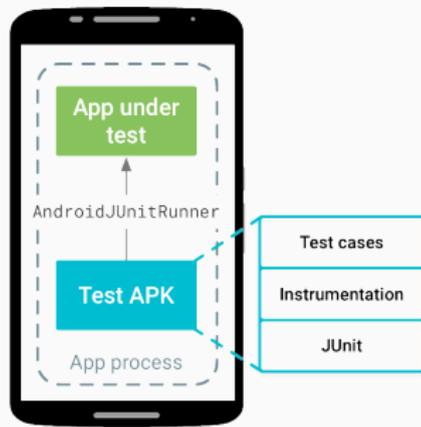
- Android supports two types of testing: **unit tests** and **instrumented tests**.
- Unit tests are located under **src/test/java**, run on the JVM, and do not have access to Android APIs.
- Instrumented tests go under **src/androidTest/java**, run on a hardware device or the emulator, and can invoke methods and modify fields in your application.
- Both types of tests are valuable, but here we will focus on instrumented tests, in particular user interface tests using the **Espresso** test framework.

<https://developer.android.com/training/testing/fundamentals.html>

Unit tests and instrumented tests



Local unit test
`src/test/java/`



Instrumented test
`src/androidTest/java/`

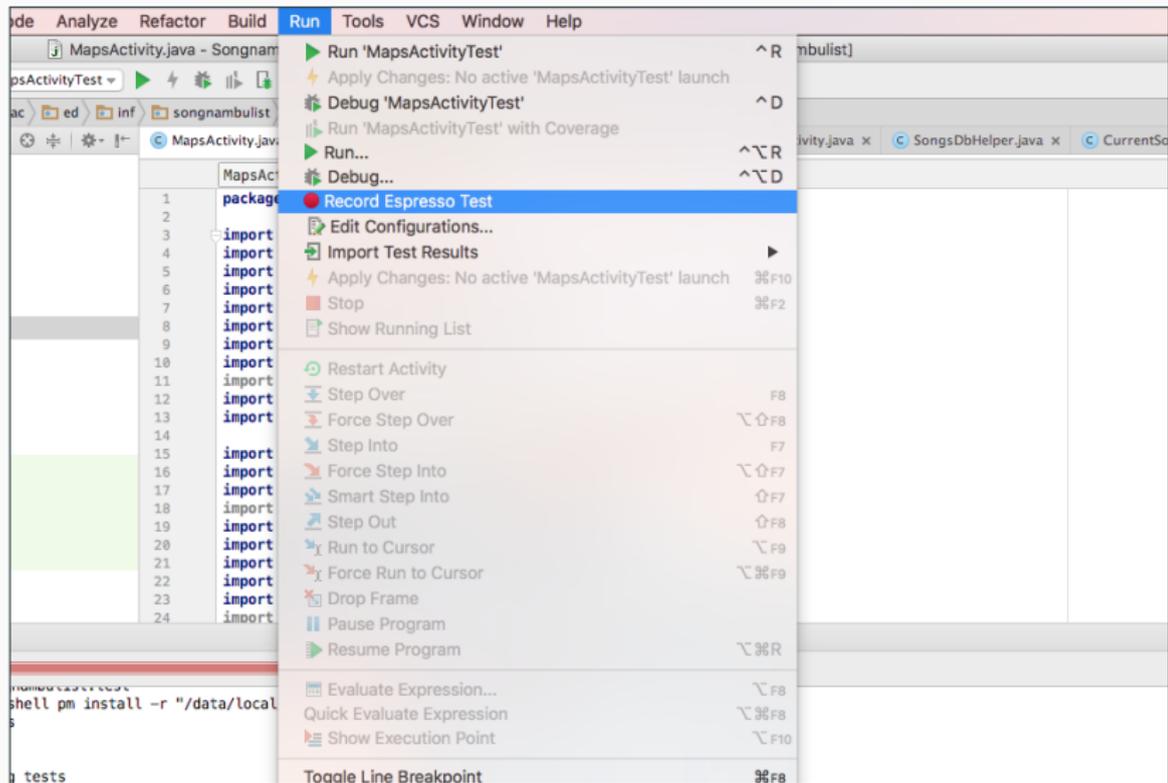
<http://developer.android.com/training/testing/start/>

Recording user interface tests

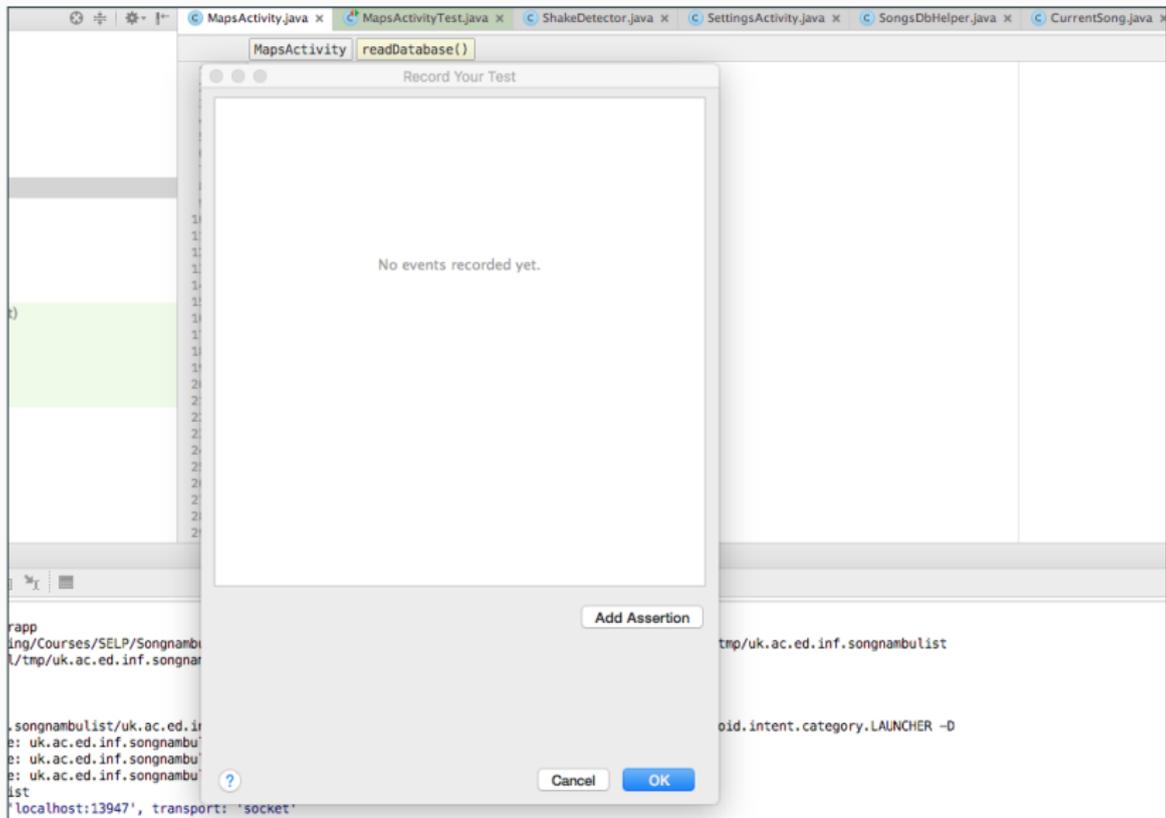
Recording user interface tests

- Android Studio provides the **Espresso Test Recorder** which tracks our interactions with our app while it is being used, and generates an Espresso test to replay these interactions in automated testing later.

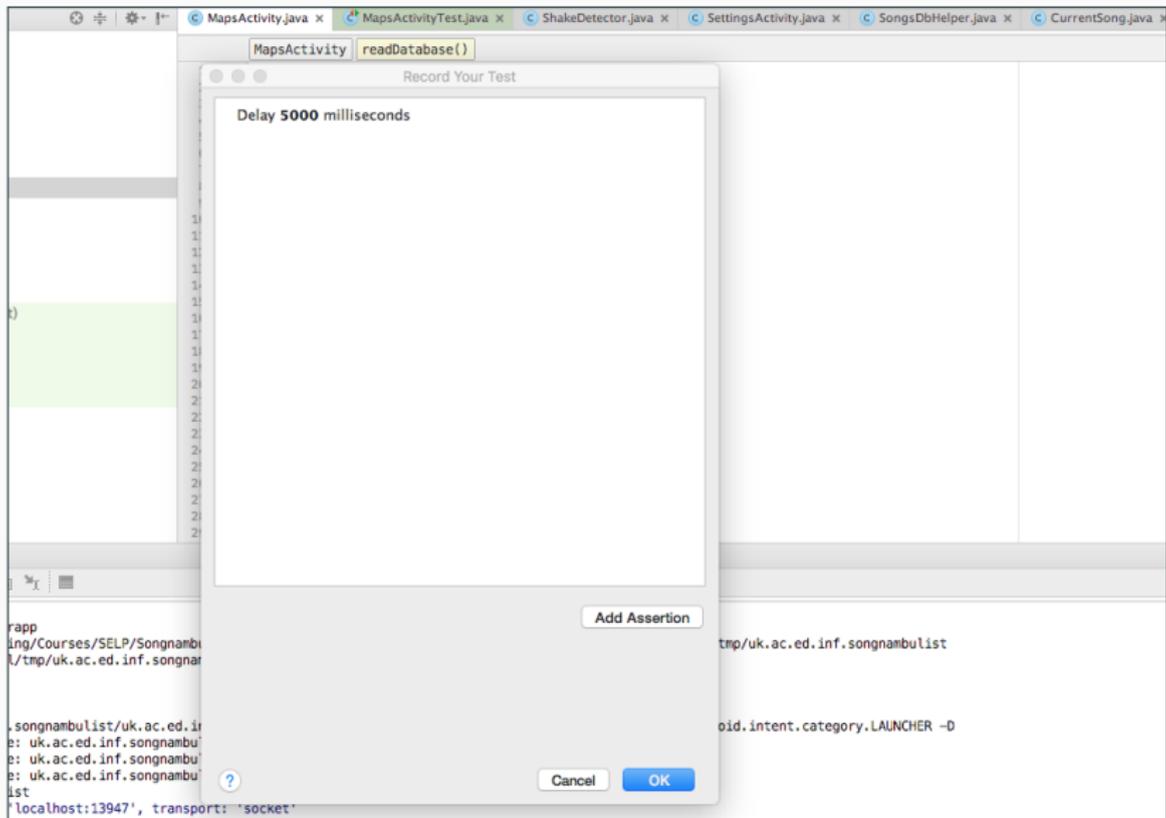
Starting the Espresso test recorder



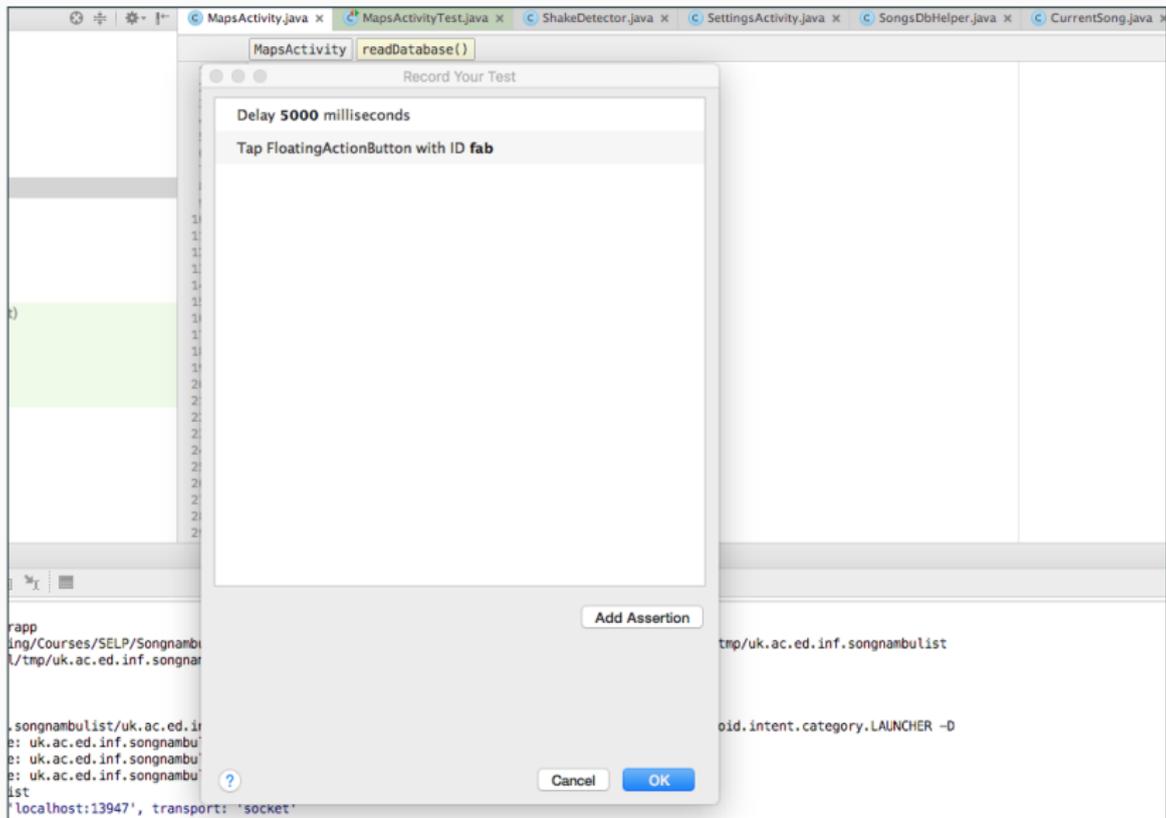
The Espresso test recorder taking notes



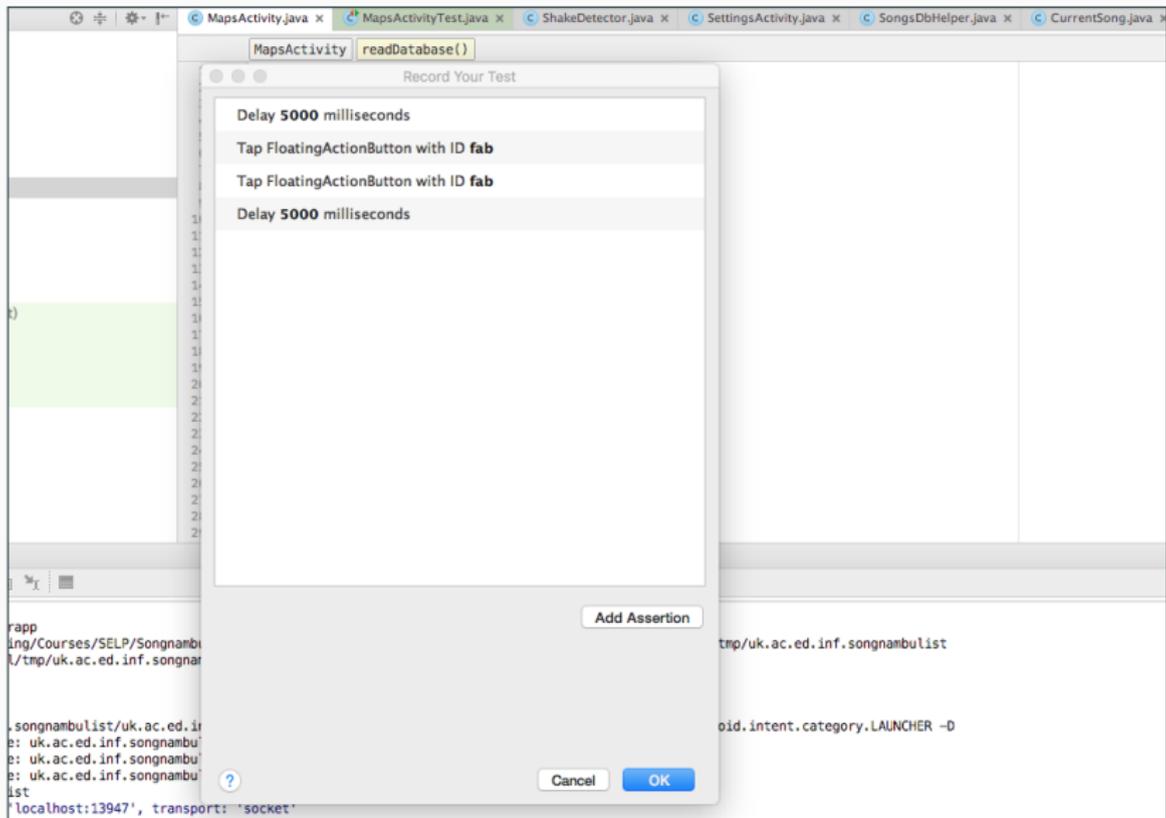
The Espresso test recorder taking notes



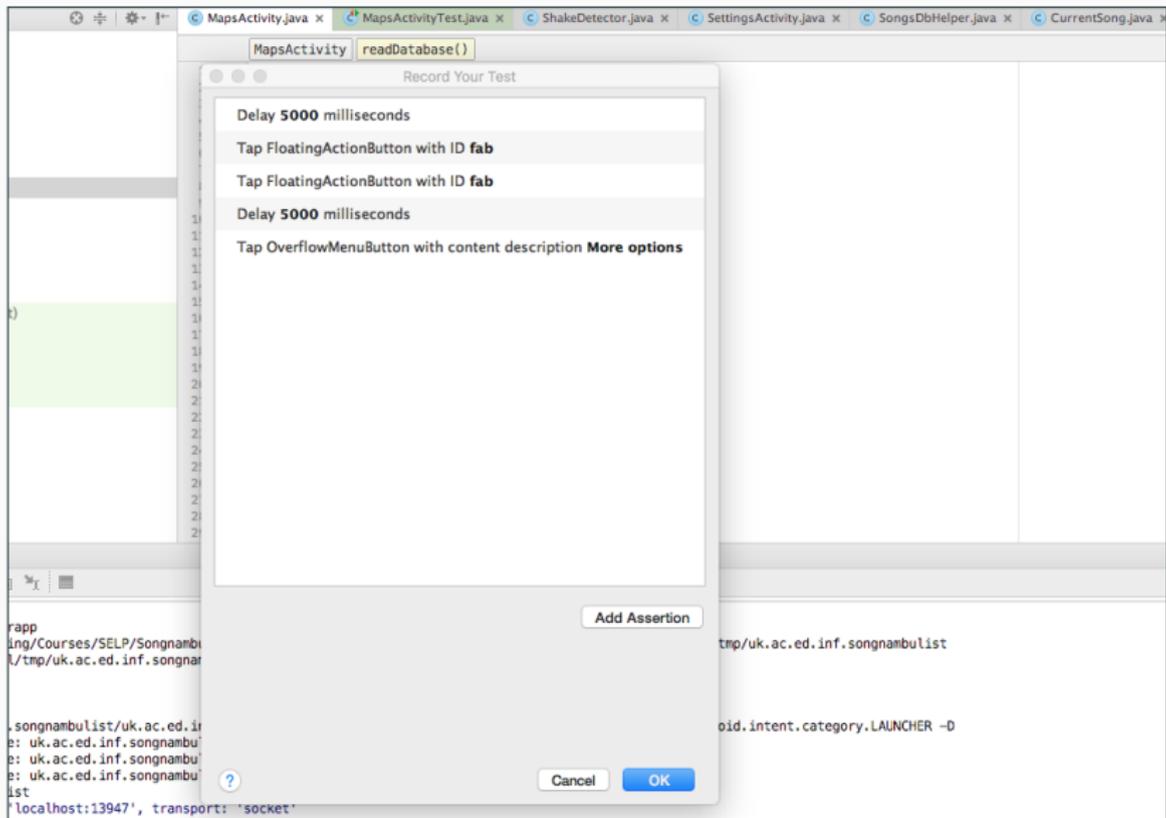
The Espresso test recorder taking notes



The Espresso test recorder taking notes



The Espresso test recorder taking notes



The code generated by the test recorder (1/2)

```
@RunWith(AndroidJUnit4.class)
public class MapsActivityTest {

    @Rule
    public ActivityTestRule<MapsActivity> mActivityTestRule = new
        ActivityTestRule<>(MapsActivity.class);

    @Test
    public void mapsActivityTest() {
        // Added a sleep statement to match the app's execution delay.
        try { Thread.sleep(5000); }
        catch (InterruptedException e) { ... }

        // First button click
        ViewInteraction floatingActionButton = onView(
            allOf(withId(R.id.fab), isDisplayed()));
        floatingActionButton.perform(click());
    }
}
```

The code generated by the test recorder (2/2)

```
// Second button click
ViewInteraction floatingActionButton2 = onView(
    allOf(withId(R.id.fab), isDisplayed()));
floatingActionButton2.perform(click());

// Added a sleep statement to match the app's execution delay.
try { Thread.sleep(5000); }
catch (InterruptedException e) { ... }

// Clicked on the overflow menu (:) in the app bar
openActionBarOverflowOrOptionsMenu(
    getInstrumentation().getTargetContext());
}
}
```

Adding unit tests

- We can then edit this test to add in JUnit assertions of the form `assertTrue`, `assertFalse`, `assertNotNull`, `assertEquals`, `assertArrayEquals`, and others.



```
import android.location.Location;
import static org.junit.Assert.*;
...
Location loc = mActivityTestRule.getActivity().getLocation();
assertTrue("Location is not null", loc != null);
```

Making our app testable [in **MapActivity**]

- We may need to add some methods to our Activity to make values visible for testing.
- We can annotate these to show that they are used for testing. The annotation **@VisibleForTesting** prevents us from actually calling this method from production code.



```
import android.support.annotation.VisibleForTesting;
...
private Location mLastLocation;
...
@VisibleForTesting
public Location getLocation() {
    return mLastLocation;
}
```

Running tests

Running tests

- Classes which contain tests are annotated with the annotation `@RunWith(AndroidJUnit4.class)` which means that they will be executed under the supervision of a **test runner**.
- This specifies the **AndroidJUnitRunner** class provided in the Android Testing Support Library as the default test runner.
- The emulator will start up as usual, but it will receive input events (such as button clicks) from our `@Test` methods.

Running tests ...

The screenshot shows an IDE interface with a project tree on the left, a code editor in the top right, and a test runner window at the bottom.

Project Tree:

- ic_notifications_black_24dp.xml
- ic_sync_black_24dp.xml
- layout
- activity_maps.xml
- menu
- mipmap

Code Editor:

```
31 // The recommended way to handle such scenarios is to use Espresso idling res
32 // https://google.github.io/android-testing-support-library/docs/espresso/idl
33
34 try {
35     Thread.sleep(5000);
36 } catch (InterruptedException e) {
37     e.printStackTrace();
38 }
```

Test Runner Window:

Running tests... Testing started at 05:47 ...

- uk.ac.ed.inf.songnambulist.MapsActi
- mapsActivityTest

```
11/01 05:47:44: Launching MapsActivityTest
No apk changes detected since last installation, skipping installation of
/Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/app-debug.apk
$ adb shell am force-stop uk.ac.ed.inf.songnambulist
No apk changes detected since last installation, skipping installation of
/Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/app-debug-androidTes
$ adb shell am force-stop uk.ac.ed.inf.songnambulist.test
Running tests

$ adb shell am instrument -w -r -e debug false -e class uk.ac.ed.inf.songnambulist.MapsActivityTest uk.ac.ed.inf
.AndroidJUnitRunner
Client not ready yet..
Started running tests
```

Bottom Bar: Run, Debug, TODO, Android Monitor, Terminal, Messages

Running tests ... [Success]

The screenshot shows an IDE interface with a project tree on the left, a code editor at the top right, and a Run window at the bottom. The Run window displays the test results for `MapsActivityTest`, indicating that the test passed successfully in 13s 11ms.

```
ic_notifications_black_24dp.xml
ic_sync_black_24dp.xml
layout
  activity_maps.xml
  menu
  mipmap
```

```
31 // The recommended way to handle such scenarios is to use Espresso idling res
32 // https://google.github.io/android-testing-support-library/docs/espresso/idl
33 try {
34     Thread.sleep(5000);
35 } catch (InterruptedException e) {
36     e.printStackTrace();
37 }
38
```

Run MapsActivityTest

1 test passed - 13s 11ms

Test Results 13s 11ms

- uk.ac.ed.inf.songnambulist.MapsActi
 - mapsActivityTest

```
11/01 05:47:44: Launching MapsActivityTest
No apk changes detected since last installation, skipping installation of
/Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/app-debug.apk
$ adb shell am force-stop uk.ac.ed.inf.songnambulist
No apk changes detected since last installation, skipping installation of
/Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/app-debug-androidTest
$ adb shell am force-stop uk.ac.ed.inf.songnambulist.test
Running tests

$ adb shell am instrument -w -r -e debug false -e class uk.ac.ed.inf.songnambulist.MapsActivityTest uk.ac.ed.inf
.AndroidJUnitRunner
Client not ready yet..
Started running tests
Tests ran to completion.
```

Run Debug TODO Android Monitor Terminal Messages

Running tests ...

The screenshot shows an IDE interface with a project tree on the left, a code editor in the center, and a terminal window at the bottom. The project tree includes files like `ic_notifications_black_24dp.xml`, `ic_sync_black_24dp.xml`, `layout`, `activity_maps.xml`, `menu`, and `miplib`. The code editor displays Java code for an Espresso test, with line numbers 31 through 38. The terminal window shows the output of a test run, including the command `adb push` and `adb shell pm install`.

```
31 // The recommended way to handle such scenarios is to use Espresso 1
32 // https://google.github.io/android-testing-support-library/docs/espresso
33 try {
34     Thread.sleep(5000);
35 } catch (InterruptedException e) {
36     e.printStackTrace();
37 }
38
```

Run MapsActivityTest

Instantiating tests...

Testing started at 05:46 ...

```
11/01 05:46:47: Launching MapsActivityTest
$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/ap
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist"
```

Build Variants

Run Debug TODO Android Monitor Terminal Messages

Running tests ...

The screenshot shows an IDE interface with a project tree on the left, a code editor in the center, and a terminal window at the bottom. The project tree includes files like `ic_notifications_black_24dp.xml`, `ic_sync_black_24dp.xml`, `layout`, `activity_maps.xml`, `menu`, and `miplibmap`. The code editor shows a Java snippet for handling exceptions with Espresso. The terminal window displays the output of running tests, including the command `adb push` and `adb shell pm install`.

```
31 // The recommended way to handle such scenarios is to use Espresso 1
32 // https://google.github.io/android-testing-support-library/docs/espresso
33 try {
34     Thread.sleep(5000);
35 } catch (InterruptedException e) {
36     e.printStackTrace();
37 }
38
```

Run MapsActivityTest

Instantiating tests...

Testing started at 05:46 ...

```
11/01 05:46:47: Launching MapsActivityTest
$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/apk
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist"
Success

$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/apk
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist.test"
```

Build Variants

Run Debug TODO Android Monitor Terminal Messages

Running tests ...

The screenshot shows an IDE interface with a code editor at the top and a Run console at the bottom. The code editor displays XML layout files and a Java test method. The Run console shows the execution of the test, including shell commands for installing APKs and running the test.

```
ic_notifications_black_24dp.xml 31
ic_sync_black_24dp.xml          32
layout                           33
  activity_maps.xml              34
  menu                           35
  mipmap                          36
  37
  38
```

```
// The recommended way to handle such scenarios is to use Espresso 1
// https://google.github.io/android-testing-support-library/docs/espresso
try {
    Thread.sleep(5000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Run MapsActivityTest

Running tests...
uk.ac.ed.inf.songnambulist.MapsAct
mapsActivityTest

Testing started at 05:46 ...

```
11/01 05:46:47: Launching MapsActivityTest
$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/apk.apk /data/local/tmp/uk.ac.ed.inf.songnambulist.apk
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist.apk"
Success

$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/apk.apk /data/local/tmp/uk.ac.ed.inf.songnambulist.test.apk
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist.test.apk"
Success

Running tests

$ adb shell am instrument -w -r -e debug false -e class uk.ac.ed.inf.songnambulist.MapsActivityTest uk.ac.ed.inf.songnambulist.MapsActivityTest
Client not ready yet..
Started running tests
```

Build Variants
Favorites

Run Debug TODO Android Monitor Terminal Messages

Running tests ... [Failure]

The screenshot shows an IDE interface with a project tree on the left, a code editor at the top right, and a Run console at the bottom. The Run console displays the following output:

```
Run MapsActivityTest
Terminated 14s 301ms
  uk.ac.ed.inf.songnambulist 14s 301ms
    mapsActivityTest 14s 301ms

Testing started at 05:46 ...

11/01 05:46:47: Launching MapsActivityTest
$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/apk.apk /data/local/tmp/uk.ac.ed.inf.songnambulist
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist"
Success

$ adb push /Users/stg/Documents/Teaching/Courses/SELP/Songnambulist/Songnambulist/app/build/outputs/apk/apk.apk /data/local/tmp/uk.ac.ed.inf.songnambulist.test
$ adb shell pm install -r "/data/local/tmp/uk.ac.ed.inf.songnambulist.test"
Success

Running tests

$ adb shell am instrument -w -r -e debug false -e class uk.ac.ed.inf.songnambulist.MapsActivityTest uk.ac.ed.inf.songnambulist.test
Client not ready yet.
Started running tests

Test failed to run to completion. Reason: 'Instrumentation run failed due to 'Process crashed.'. Check de
Test running failed: Instrumentation run failed due to 'Process crashed.'
```

The code editor shows the following Java code snippet:

```
31 // The recommended way to handle such scenarios is to use Espresso 1
32 // https://google.github.io/android-testing-support-library/docs/espresso
33 try {
34     Thread.sleep(5000);
35 } catch (InterruptedException e) {
36     e.printStackTrace();
37 }
38
```

When tests fail

- Tests (especially tests generated by the Espresso test recorder) can fail for reasons other than an error in our application logic. It is important to look at the reason why the test failed; it might be a poorly-specified test.
- **False positive** failures can be caused by timing issues where the app under test does not respond within the delay anticipated by the **sleep** pause in the test.
- Work is underway to improve the Android testing framework.

Links

- <https://developer.android.com/training/testing/index.html>
- <https://developer.android.com/training/testing/fundamentals.html>
- <https://developer.android.com/training/testing/junit-rules.html>
- <https://developer.android.com/training/testing/espresso/index.html>
- <https://developer.android.com/studio/write/annotations.html>