Android testing

Group members: Ovidiu Floca & Mohamed Ali Abubakar

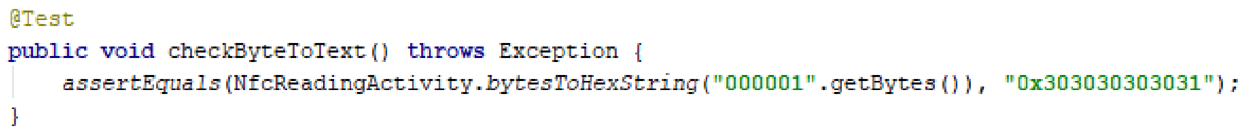
Exploring the difference in Android testing between tests that can be run on the JVM and the ones that require a device to be tested

In Android, there are two types of tests

- Tests that can be run on the local java virtual machine and that don't require the Android APIs.
  This kind of test is efficient and fast because it doesn't require the test code to be loaded onto a device or the emulator every time it is run.
  In order to be able to run unit tests on Android, the dependency has to be imported in Gradle:

```
dependencies {
    // Required -- JUnit 4 framework
    testCompile 'junit:junit:4.12'
}
```

  All the test files have to be placed inside the /src/test/java folder, which is automatically created by Android Studio with every new project.
  Android unit tests are identical to normal Java unit tests and have the same structure:



```
@Test
public void checkByteToText() throws Exception {
    assertEquals(NfcReadingActivity.bytesToHexString("000001".getBytes()), "0x303030303031");
}
```

  The previous test checks if the method bytesToHexString from NfcReadingActivity correctly converts a set of bytes into hex.

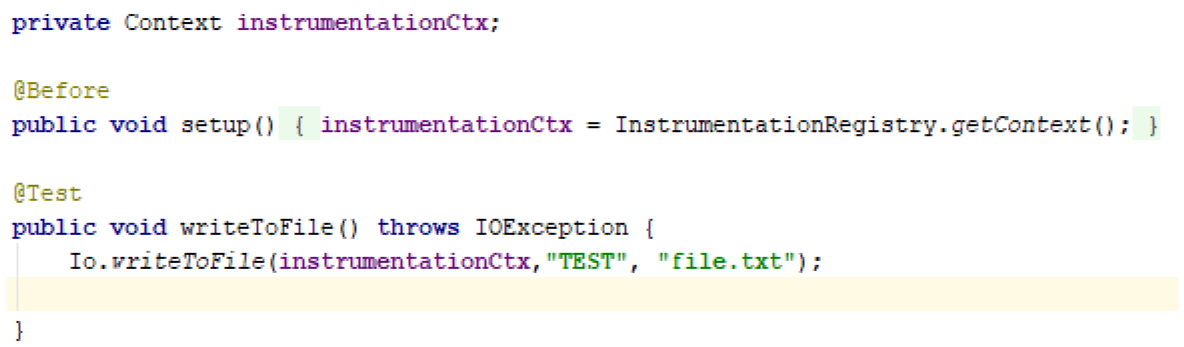
- Tests that need the Android APIs to run.

  This kind of test requires a device or the emulator to run and should only be used when the test requires an Android component, like the Context.

  In order to set it up, the following dependencies have to be added in Gradle:

```
dependencies {
    androidTestCompile 'com.android.support:support-annotations:25.3.1'
    androidTestCompile 'com.android.support.test:runner:0.5'
    androidTestCompile 'com.android.support.test:rules:0.5'
}
```

All the test files have to be placed inside the /src/androidTest/java folder, which is automatically created by Android Studio with every new project.



```
private Context instrumentationCtx;

@Before
public void setup() { instrumentationCtx = InstrumentationRegistry.getContext(); }

@Test
public void writeToFile() throws IOException {
    Io.writeToFile(instrumentationCtx,"TEST", "file.txt");

}
```

The previous test checks if the writeToFile method works. The method requires a not null context in order to create the file and in order to have it, we declare it and set it up in the setup() method.

Explore developing automated UI tests for Android using the Espresso framework

The purpose of UI testing is to ensure that the users don't run into problems when using the application.

In order to set the app up for Espresso testing, the following dependency has to be added to Gradle:

```
dependencies {
    androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'
}
```

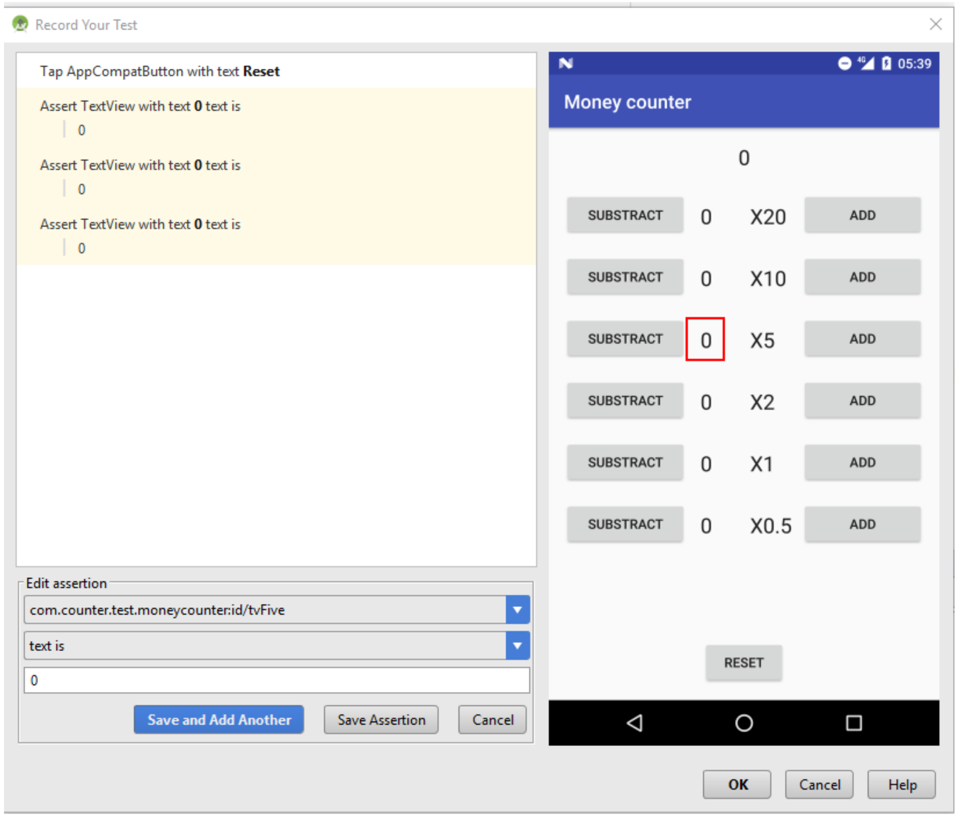
Espresso tests are placed inside the /src/androidTest/java folder.

```
@Rule
public ActivityTestRule<MainActivity> mActivityRule =
        new ActivityTestRule<>(MainActivity.class);

@Test
public void ensureAddTwentyWorks() {
    String text = getText(withId(R.id.tvTwenty));
    int amount = Integer.parseInt(text);
    amount++;
    onView(withId(R.id.bAddTwenty)).perform(click());
    onView(withId(R.id.tvTwenty)).check(matches(withText(""+amount)));
}
```

The previous test checks that a TextView changes its text when a specific button is pressed.

Experiment with the Espresso test recorder in order to create UI tests without coding them


Record Your Test

Tap AppCompatButton with text **Reset**

Assert TextView with text **0** text is
0

Assert TextView with text **0** text is
0

Assert TextView with text **0** text is
0

Edit assertion
com.counter.test.moneycounter:id/tvFive
text is
0

Using the Espresso test recorder in order to create UI tests is really easy. It’s only a matter of pressing a few buttons and the Espresso test class is created automatically.

Reflections

The fact that Android is written in Java makes using JVM unit tests easy to run because of the familiarity from writing normal Java programs.

Instrumented tests on the other hand are a bit more difficult to implement because of the necessity to use the Android API and the documentation is a bit scarcer than for normal jUnit tests.

Espresso testing is really intuitive to set up and use and using the documentation provided by the Android team makes using Espresso testing a must-do for every Android project as it provides a great insight into how the users use the app and the problems that they might face.

To makes things even simpler for the developers, which in general don't do a lot of test implementation in their projects, the Android team created the Espresso test recorder.

After analyzing the usefulness and ease of implementation of testing in Android, we will make sure that all our future projects will have both unit and UI tests.