BDD testing on Android using Green Coffee and Espresso

Green Coffee is a library that allows you to run your acceptance tests written in Gherkin language in your Android instrumentation tests using the step definitions that you declare.
Gherkin is a Business Readable, Domain Specific Language created especially for behaviour descriptions. It gives you the ability to remove logic details from behaviour tests. Gherkin uses indentation to define structure. Line endings terminate statements (called steps) and either spaces or tabs may be used for indentation.

Step-by-step guide to install and test an app

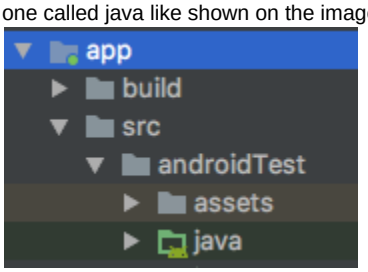
1. Open the Project using Android Studio.
2. To the build.gradle app file we will add this lines to the dependencies.

```
androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation
'com.android.support.test:rules:1.0.2'
    androidTestImplementation
'com.mauriciotogneri:greencoffee:3.5.0'
```

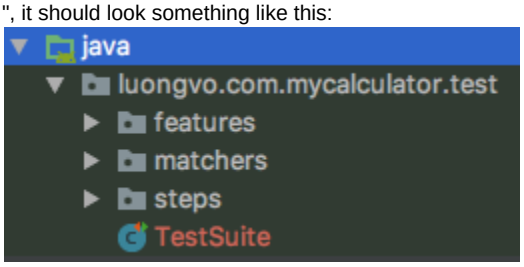
   And the debug variable to the buildTypes

```
debug {
            testCoverageEnabled true
        }
```

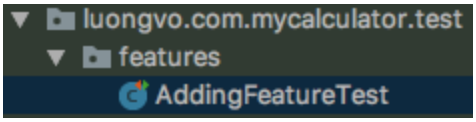
   This will sync our project with all the dependecies we need to run the tests
3. In the project tab, go to /app/src and create a new folder called androidTest, inside this folder create a new folder called assets and other one called java like shown on the image:



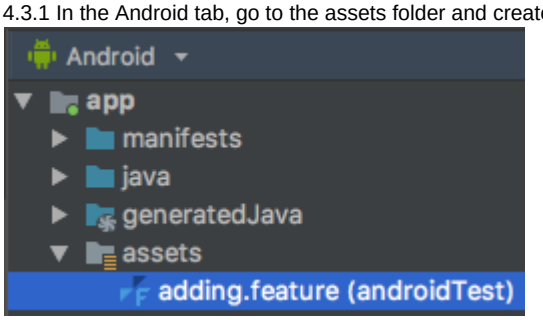
   In the assets folder we will write the Gherkin code.
4. Inside the java folder create a new folder called "com.application.name.test" (change the com.application.name to your project's name) and inside this folder we will create three new folders called "**features**", "**matchers**" and "**steps**"; also create a file called "**TestSuite.java** ", it should look something like this:



   The "**TestSuite.java**" file will be the one to run when testing; it will contain the declaration of the Feature files we create and also the asignation of the local language to be used E.g English or Spanish.
   4.1 To the features folder we add a new with a name related to what we want to test. For example:



This file will contain a class that extends from "**GreenCoffeeTest"**and also will declare the Activity, the feature and the step definitions that will be used.
4.2 With the feature definitions created, we will need to create as well the steps definitions and the features.
4.3 Let's write the feature.
4.3.1 In the Android tab, go to the assets folder and create a new .feature file:



5. The Behaviour Driven Testing assumes that we create scenarios that we'd like to test and then write the test code. A typical scenario written using the Gherkin language looks like:

```
Feature: Login feature

  Scenario: As a valid user I can log into my app
    When I press "Login"
    Then I see "Welcome to coolest app ever"
```

Every scenario consists of a list of steps, which must start with one of the keywords Given, When, Then, But or And, for example:

```
Feature: Use the calculator
 Scenario: adding two numbers
     When I press the 1 button
     And I press the + button
     And I press the 2 button
     And I press the = button
     And I see the number 3 written
     Then I press the clear button
```

To read further on the Gherkin documentation go to this URL: https://docs.cucumber.io/gherkin/reference/
So we will write a test on the .feature file stored on the assets folder:

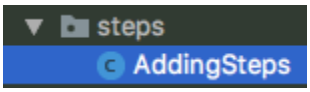
```
Feature: Use the calculator

 Background:
  Given I see an empty calculator

 Scenario: button is pressed
  When I press the 1 button
  Then I see the number 1 written

 Scenario: adding two numbers
     When I press the 1 button
     And I press the + button
     And I press the 2 button
     And I press the = button
     And I see the number 3 written
     Then I press the clear button
```

6. With the test written with the Gherkin language we need to define the steps that we want to use so Espresso can recognize what we need to do:
   6.1 Create a new file on the steps folder, like so:



To this file we will write the definitions for what we need to test and what does the user wants to do.

```
public class AddingSteps extends GreenCoffeeSteps {


    @Given("^I see an empty calculator$")
    public void iSeeAnEmptyLoginForm() {
        onViewWithId(R.id.text_view_result).isEmpty();
    }

    @When("^I press the (\\d|\\+|\\-|\\*|\\/|=|clear) button$")
    public void iPressThe1Button(String button) {
        switch (button) {
            case "1":
                onViewWithId(R.id.button1).click();
                break;
            case "2":
                onViewWithId(R.id.button2).click();
                break;
            case "3":
                onViewWithId(R.id.button3).click();
                break;
            case "4":
                onViewWithId(R.id.button4).click();
```

```
                break;
            case "5":
                onViewWithId(R.id.button5).click();
                break;
            case "6":
                onViewWithId(R.id.button6).click();
                break;
            case "7":
                onViewWithId(R.id.button7).click();
                break;
            case "8":
                onViewWithId(R.id.button8).click();
                break;
            case "9":
                onViewWithId(R.id.button9).click();
                break;
            case "0":
                onViewWithId(R.id.button0).click();
                break;
            case "+":
                onViewWithId(R.id.buttonAdd).click();
                break;
            case "-":
                onViewWithId(R.id.buttonSub).click();
                break;
            case "*":
                onViewWithId(R.id.buttonMul).click();
                break;
            case "/":
                onViewWithId(R.id.buttonDiv).click();
                break;
            case "=":
                onViewWithId(R.id.buttonEqual).click();
                break;
            case "clear":
                onViewWithId(R.id.buttonClear).click();
                break;
            default:
                throw new RuntimeException();
        }
    }


    @Then("^I see the number (\\d+) written$")
    public void iSeeTheNumberWritten(String number) {

onViewWithId(R.id.text_view_result).check(ViewAssertions.matches(Vi
ewMatchers.withText(number)));
```

```
    }

}
```

Using regular expressions and the Espresso GUI utilities we can create step definitions that are reusable, easy to understand and simple to write.

7. After we create the test and define what each step means we need to reference them in the **"AddingFeatureTest.java",** this file will join the Activity with the test feature and the steps definitions:

```
@RunWith(Parameterized.class)public class AddingFeatureTest extends
GreenCoffeeTest {
    @Rule
    public ActivityTestRule<MainActivity> activity = new
ActivityTestRule<>(MainActivity.class);

    public AddingFeatureTest(ScenarioConfig scenarioConfig)
    {
        super(scenarioConfig);
    }

    @Parameters(name = "{0}")
    public static Iterable<ScenarioConfig> scenarios() throws
IOException {
        return new GreenCoffeeConfig()
                .withFeatureFromAssets("assets/adding.feature")
                .takeScreenshotOnFail()
                .scenarios(ENGLISH, SPANISH);
    }

    @Test
    public void test() {
        start(new AddingSteps());
    }

    @Override
    protected void beforeScenarioStarts(Scenario scenario, Locale
locale) {
        // do something
    }
}
```

8. Then with the Feature referenced in the "**AddinfFeatureTest.java"** we will define the "**TestSuite.java"** wich will be what we select in order to run all the test:

```
@RunWith(Suite.class)@Suite.SuiteClasses({
        AddingFeatureTest.class,
})
public class TestSuite
{
    public static final Locale ENGLISH = new Locale("en", "GB");
    public static final Locale SPANISH = new Locale("es", "ES");
}
```

9. In case that comparing objects its needed, is useful to create a file called matcher, in here we will write how we need to compare the object. For example if we need to compare two Person objects, we can do so by their names.

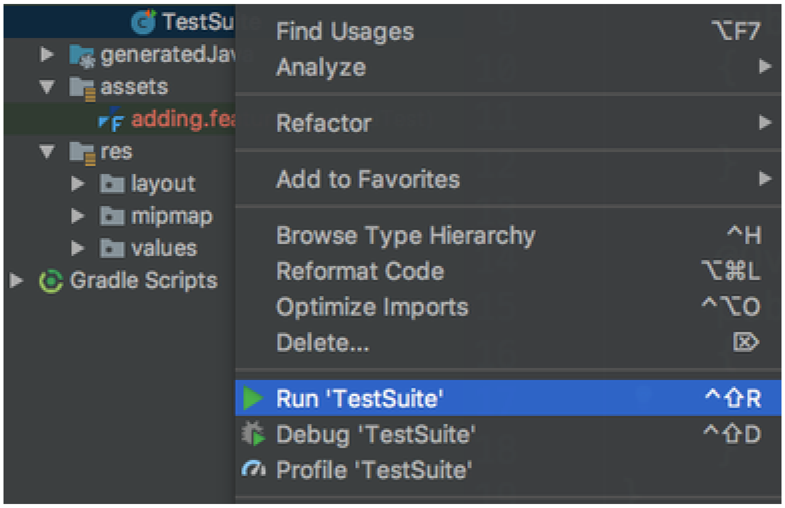
```
public class PersonMatcher extends DataMatcher<Contact, String>
{
    public PersonMatcher(int resourceId)
    {
        super(resourceId, Person.class);
    }

    @Override
    public boolean matches(Contact contact, String content)
    {
        return person.name().equals(content);
    }
}
```

And then call this matcher to the step definitions file to know if we received the necessary data or not. Like so:

```
@When("^I select the contact called '([\\w| ]+)'$")
    public void iSelectTheContactCalled$(String username)
    {
        DataMatcher<Contact, String> dataMatcher = new
PersonMatcher(R.id.contacts_list);
        dataMatcher.with(username).click();
    }
```

10. With all the test written and the steps defined we can run the test by going to the TestSuite file, right click and Run 'TestSuite'

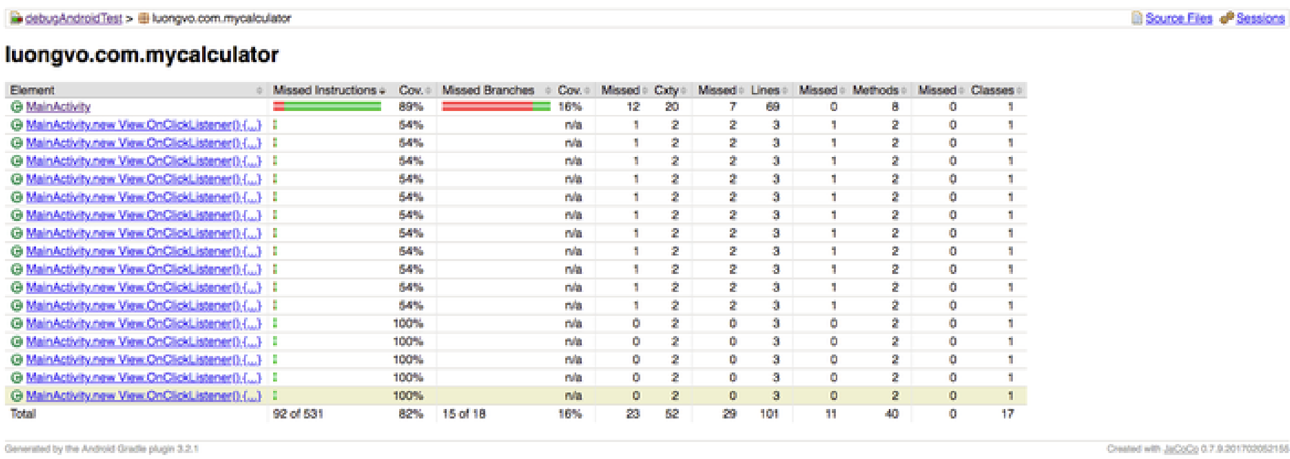


The app will install on the device you select and the test will run

11. To run the code coverage test use the command:

```
./gradlew createDebugCoverageReport
```

After this the code coverage report will be generated in `app/reports/coverage/debug` .
This is how the code coverage reports will look:


debugAndroidTest > luongvo.com.mycalculator　　Source Files　Sessions

luongvo.com.mycalculator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MainActivity		89%		16%	12	20	7	69	0	8	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		54%		n/a	1	2	2	3	1	2	0	1
MainActivity.new View.OnClickListener() {...}		100%		n/a	0	2	0	3	0	2	0	1
MainActivity.new View.OnClickListener() {...}		100%		n/a	0	2	0	3	0	2	0	1
MainActivity.new View.OnClickListener() {...}		100%		n/a	0	2	0	3	0	2	0	1
MainActivity.new View.OnClickListener() {...}		100%		n/a	0	2	0	3	0	2	0	1
MainActivity.new View.OnClickListener() {...}		100%		n/a	0	2	0	3	0	2	0	1
Total	92 of 531	82%	15 of 18	16%	23	52	29	101	11	40	0	17

Generated by the Android Gradle plugin 3.2.1　　Created with JaCoCo 0.7.9.201702052155

debugAndroidTest > luongvo.com.mycalculator > MainActivity — Sessions

MainActivity

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
onEqualButtonClicked()		55%		16%	12	13	7	17	0	1
initControlListener()		100%		n/a	0	1	0	17	0	1
initControl()		100%		n/a	0	1	0	18	0	1
onNumberButtonClicked(String)		100%		n/a	0	1	0	4	0	1
onOperatorButtonClicked(String)		100%		n/a	0	1	0	4	0	1
onCreate(Bundle)		100%		n/a	0	1	0	5	0	1
onClearButtonClicked()		100%		n/a	0	1	0	3	0	1
MainActivity()		100%		n/a	0	1	0	1	0	1
Total	37 of 357	89%	15 of 18	16%	12	20	7	69	0	8

Generated by the Android Gradle plugin 3.2.1 — Created with JaCoCo 0.7.9.201702052155

```
28.     Button buttonEqual;
29.     String result;
30.     String tmp;
31.     String operator;
32.     TextView resultTextView;
33.
34.
35.     @Override
36.     protected void onCreate(Bundle savedInstanceState) {
37.         super.onCreate(savedInstanceState);
38.         setContentView(R.layout.activity_main);
39.         initControl();
40.         initControlListener();
41.     }
42.
43.     private void initControlListener() {
44.         button0.setOnClickListener(new View.OnClickListener() {
45.             @Override
46.             public void onClick(View v) {
47.                 onNumberButtonClicked("0");
48.             }
49.         });
50.         button1.setOnClickListener(new View.OnClickListener() {
51.             @Override
52.             public void onClick(View v) {
53.                 onNumberButtonClicked("1");
54.             }
```

12. For an example to a project you can clone this repository: https://github.com/vndly/green-coffee-example or download this other simplified example of a calculator:

13. For extra information about Gherkin, cucumber, green coffie visit this links:
    http://www.methodsandtools.com/tools/greencoffee.php
    https://bitbar.com/guest-blog-how-to-use-gherkin-and-espresso-for-android-test-automation/
    http://docs.behat.org/en/v2.5/guides/1.gherkin.html#thens
    https://medium.com/@Alex_Fogleman/into-the-belly-of-the-beast-ui-testing-an-android-app-with-cucumber-and-espresso-4a65ae61539a
    https://docs.cucumber.io/guides/10-minute-tutorial/
    https://proandroiddev.com/be-da-developer-cucumber-on-android-cfd07773e59d
    https://medium.com/@thexap/how-to-automate-testing-an-android-react-native-app-with-calabash-and-cucumber-bf12bfbd3c8
    http://overtest.me/android-bdd-tests-part-2-time-for-cucumber/
    https://github.com/calabash/calabash-ios/wiki/Getting-Started
    https://github.com/calabash
    https://developer.xamarin.com/guides/testcloud/calabash/configuring/osx/#
    https://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html