Introduction

we'll take a look at Spock, a Groovy testing framework. Mainly, Spock aims to be a more powerful alternative to the traditional JUnit stack, by leveraging Groovy features.

By making use of Groovy, Spock introduces new and expressive ways of testing our Java applications, which simply aren't possible in ordinary Java code. We'll explore some of Spock's high-level concepts during this article, with some practical step-by-step examples.

Prerequisite

You have some knowledge about spring boot and testing concept mock.

For installing Spock into the spring boot

Add this dependency into pom.xml

```
<dependency>
   <groupId>org.spockframework</groupId>
   <artifactId>spock-core</artifactId>
   <version>2.3-groovy-4.0</version>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.codehaus.groovy</groupId>
   <artifactId>groovy-all</artifactId>
   <version>2.4.21</version>
   <scope>test</scope>
</dependency>
```

And add a plugin for groovy support

```xml
<plugin>
   <groupId>org.codehaus.gmavenplus</groupId>
   <artifactId>gmavenplus-plugin</artifactId>
   <version>1.5</version>
   <executions>
       <execution>
           <goals>
               <goal>compile</goal>
               <goal>testCompile</goal>
           </goals>
       </execution>
   </executions>
</plugin>
```

Now for making a straightforward test

Make groovy class and extend specification (spock.lang.specification).

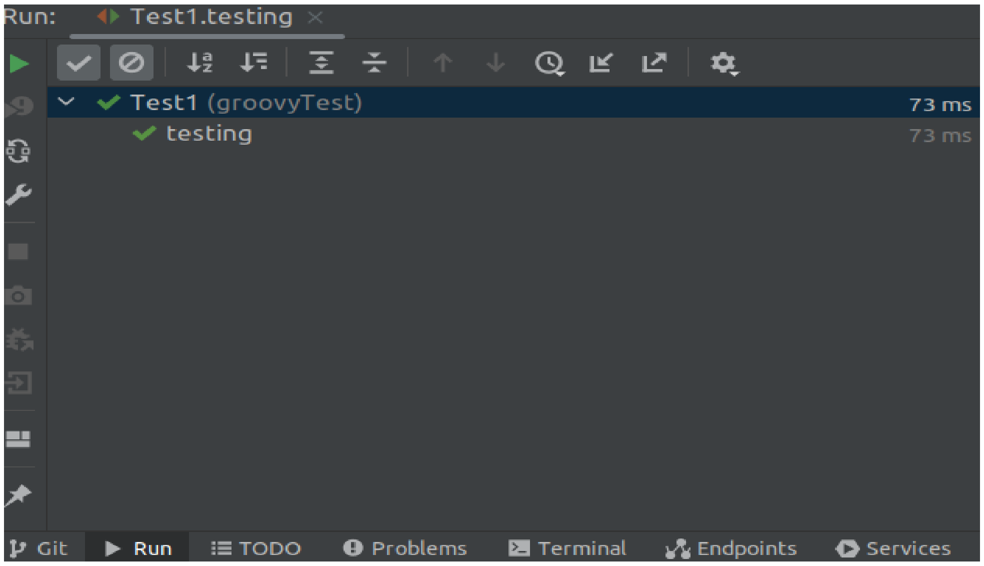
```groovy
@SpringBootTest
class Test1 extends Specification {

}
```

For testing purposes add one method

```groovy
def "testing"(){

   expect:

   1+1 == 2

}
```

Now run the test

Output:
You see Test Case Passed successfully.



```
Test1 (groovyTest)     73 ms
  testing              73 ms
```

Controller Test-cases

After Simple Test Case We are going to do Controller Testing.

1)Hello world Example

First, create a ‘Hello’ controller

```
@RestController
public class Hello {

   @GetMapping("/hello")
   public String greetings(){
       return "Hello World! ";
   }

}
```

Now for this controller, we are writing test cases using Spock.

```
class HelloTest extends Specification {
   @MockBean
   def cont = new Hello()

   MockMvc mvc =
MockMvcBuilders.standaloneSetup(cont).build()

   def "when get is performed then the response has
status 200 and content is 'Hello world!'"() {
       expect: "Status is 200 and the response is
'Hello world!'"

mvc.perform(MockMvcRequestBuilders.get("/hello"))
               .andExpect(status().isOk())
               .andReturn()
               .response
               .contentAsString == "Hello World! "
   }
}
```

Here we use the **‘expect’ keyword** for testing.

Same this way we have **other keyword pairs**
Given-when-then
Given = given values or for the particular method you want to add value
When = it’s for calling methods or testing scenarios
Then = show the result

To understand this we are taking two strings into the **given** block.
In **when** block we convert these strings to lowercase and in **then block** we are doing a simple assertion check if both the string is the same or not

In real-time projects in **Given Block,** we provide objects for services and **mocking-stubbing** is also done in this block.
When Block Make the call for methods or conditions for which we test our method.
And **Then Block** Check for assertions or results if the condition's output is desired output or not.

Taken Simple Example to Compare to Strings to show **given-when-then** implementation

```
def "check case-insensitive equality of 2 strings"() {
   given: "Given Two Input Strings"
       String str1 = "hello"
       String str2 = "HELLO"

   when: "Convert String to Lowercase"
       str1 = str1.toLowerCase()
       str2 = str2.toLowerCase()

   then: "Asser it for Equality"
       str1 == str2
}
```

Important Functions

Here We Have 4 Methods used to define global or redundant code.

```
/**
* Call only one time for all test cases
* you can define a spec for static or whole test cases
that have the same value
* */
def setupSpec() {
}
/**
* call only once at the end of all test cases
* */
def cleanupSpec() {
}
/**
* call for every test case
* it calls for every test case before the test case
* */
def setup() {
}
```

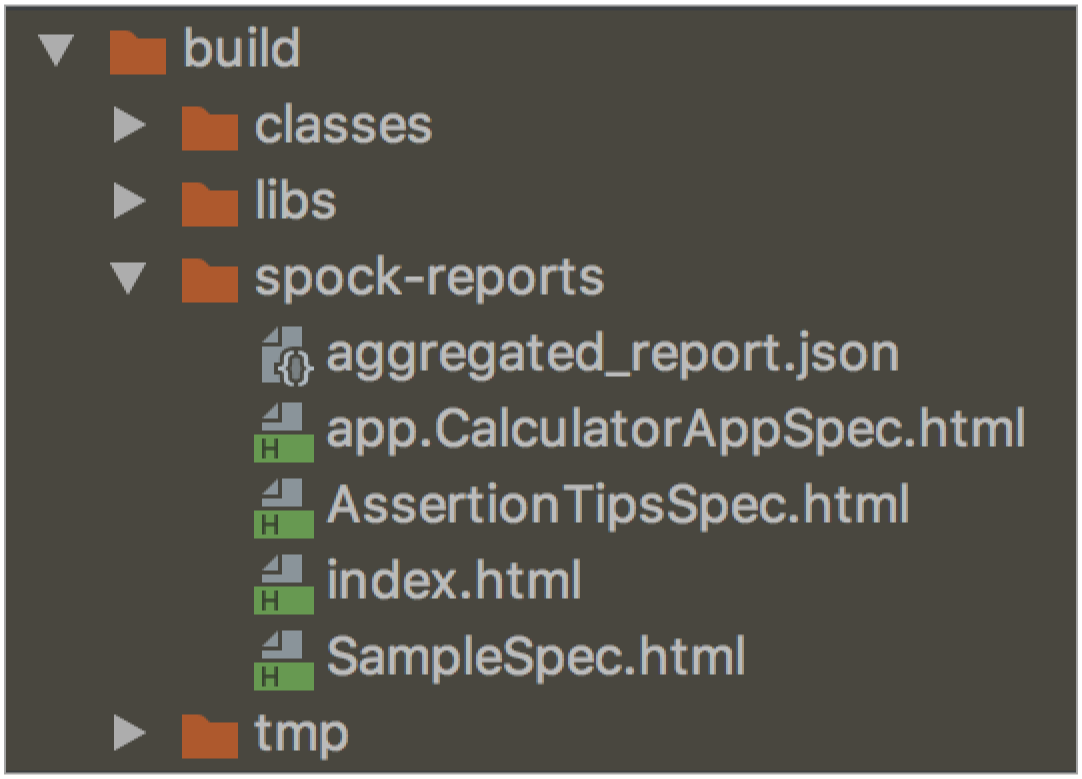
```
/**
* call for every test case
* it calls for every test case after test case
* */
def cleanup() {
}
```

Using Spock You can generate reports too
For that, we have to add 3 dependencies into pom.xml

```
<dependency>
   <groupId>com.athaydes</groupId>
   <artifactId>spock-reports</artifactId>
   <version>2.3.0-groovy-3.0</version>
   <scope>test</scope>
</dependency>

<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-api</artifactId>
   <version>2.0.6</version>
</dependency>
<dependency>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-simple</artifactId>
   <version>2.0.6</version>
   <scope>test</scope>
</dependency>
```

After that reload the maven file and run all the test cases

Then build -> spock-reports -> index.html you can find reports for test cases you run.
(For Reference you see the below image.)



```
build
  classes
  libs
  spock-reports
    aggregated_report.json
    app.CalculatorAppSpec.html
    AssertionTipsSpec.html
    index.html
    SampleSpec.html
  tmp
```

Mocking and Stubbing

For mocking you have to use the Mock() method

```
def studentDatabase = Mock(StudentDatabase.class)
```

With Mocks, you don't have to do a lot of setups, but you can validate the interactions that happened with the mock objects supplied to the application under test.

With mocks, you can do things like

- What arguments the mocks were called?
- What was the total count of invocations etc?
- Ascertaining the order of mocks.

For a mock example refer to the GitHub link.
Link = https://github.com/Ritish34/spock-demo

Stubbing is nothing but setting up pre-defined or canned responses on the Mock invocations to test the different flows/scenarios of the application under test.

A Stub is like a Mock which in a way emulates the behaviour of the real object. You can simply call it a programmed Mock.

Syntax:
StubbedObject.StubbedMethod(argument list) >> “Stubbed Response”