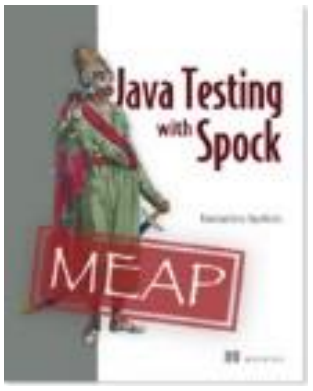
Comparing Spock and Junit

By Konstantinos Kapelonis, *Java Testing with Spock*



Spock is also a superset of the defacto testing framework for Java: Junit. In this article, excerpted from Java Testing with Spock, we will compare Spock with Junit.

In the Java world, there has been so far only one solution for unit tests. The venerable JUnit framework is the obvious choice and has become almost synonymous with unit testing. JUnit has the largest mind share among developers who are entrenched in their traditions and don't want to look any further.

Even TestNG which has several improvements and is also fully compatible with JUnit has failed to gain significant traction.

But fear not! A new testing solution is now available in the form of Spock. Spock is a testing framework written in Groovy but able to test both Java and Groovy code. It is fully compatible with JUnit (it actually builds on top of the JUnit runner) and provides a cohesive testing package that also includes mocking/stubbing capabilities

It is hard to compare JUnit and Spock in a single article, because both tools have a different philosophy when it comes to testing. JUnit is a Spartan library that provides the absolutely necessary thing you need to test and leaves additional functionality (such as mocking and stubbing) to external libraries.

Spock has a holistic approach, providing you a superset of the capabilities of JUnit, while at the same time reusing its mature integration with tools and developments environments. Spock can do everything that JUnit does and more, keeping backwards compatibility as far as test runners are concerned.

What follows is a brief tour of some Spock highlights.

Writing concise code with Groovy syntax

Spock is written in Groovy which is less verbose than Java. This means that Spock tests are more concise than the respective JUnit tests. Of course this advantage is not specific to Spock itself. Any other Groovy testing framework would probably share this trait. But at the moment only Spock exists in the Groovy world.

Here is the advantage in a visual way, shown in figure 1.

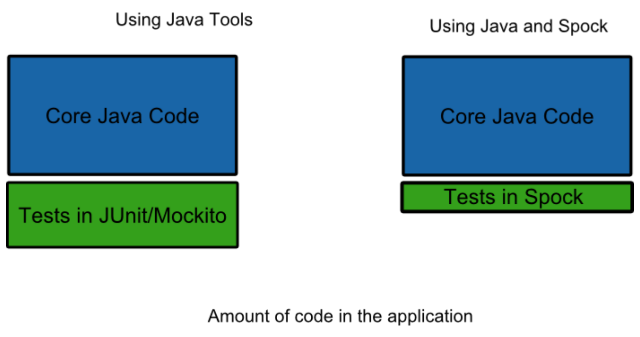


Figure 1 Amount of code in an application with JUnit and Spock tests

Less code is easier to read, easier to debug, and easier to maintain in the long run.

Mocking and Stubbing with no external library

JUnit does not support Mocking and Stubbing on its own. There are several Java framework that fill this position. This is the main reason that I got interested in Spock in the first place is the fact that it comes full batteries included as mocking and stubbing are supported out of the box.

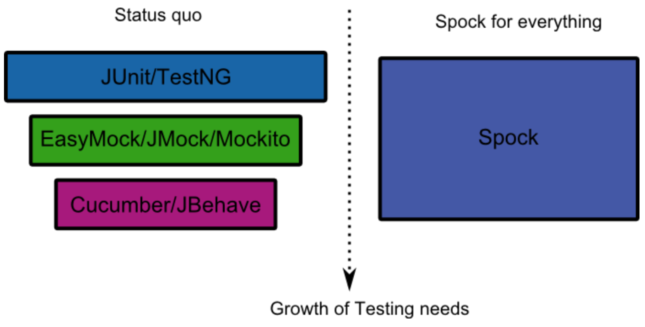


Figure 2 Spock is a superset of JUnit

I'll let this example explain:

David goes into a software company and starts working on an existing Java code base. He's already familiar with JUnit (defacto testing framework for Java). While working on the project, he needs to write some unit tests that need to run in specific order. JUnit does not support this. So David also includes TestNG in the project.

Later he realizes that he needs to use mocking for some very special features of the software (for example the credit card billing module). He spends some time to research all the available Java libraries (there are many). He chooses Mockito, and integrates it in the code base as well.

Months pass and David learns all about Behavior-Driven Development in his local Dev Meeting. He gets excited! Again he researches the tools and selects JBehave for his project in order to accomplish BDD.

Meanwhile Jane is a junior developer that knows only vanilla Java. She joins the same company and gets overwhelmed the first day because she has to learn 3-4 separate tools just to understand all the testing code.

In an alternate universe David starts working with Spock as soon as he joins the company. Spock has everything he needs for all testing aspects of the application. He never needs to add another library or spend time researching stuff as the project grows.

Jane joins the same company in this alternate universe. She asks David for hints on the testing code and he just replies "Learn Spock and you will understand all testing code". Jane is happy because she has to focus on a single library instead of three.

Even though Spock does not offer a full featured BDD workflow (as JBehave), it still offers the capability to write tests understandable by business analysts as shown in the next section.

Using English sentences in Spock tests and reports

Here is a bad JUnit test (I see these all the time). It contains cryptic method names that do not describe what is being tested.

Listing 1.7 A JUnit test where method names are unrelated to business value

```
public class ClientTest {
     @Test
     public void scenario1() #A
     {
             CreditCardBilling billing = new CreditCardBilling();
             Client client client = new Client();
             billing.chargeClient(client,150);
             assertTrue("expect bonus",client.hasBonus()); #B
     }
     @Test
     public void scenario2() #A
```

```
    {
            CreditCardBilling billing = new CreditCardBilling();
            Client client client = new Client();
            billing.chargeClient(client,150);
            client.rejectsCharge();
            assertFalse("expect no bonus",client.hasBonus());
    }
```

#A A test method with a generic name
#B Non technical people cannot understand test

This code is only understandable by programmers. Also if the second test breaks, a project manager (PM) will see the report and know that "scenario2" is broken. This report has no value for the PM since he does not know what scenario2 does exactly without looking at the code.

Spock supports an English like flow. Compare the same example in Spock:

Listing 1.8 A Spock test where methods explain the business requirements

```
class BetterSpec extends spock.lang.Specification{

    def "Client should have a bonus if he spends more than 100 dollars"() {
            when: "a client buys something with value at least 100" #A
            def client = new Client();
            def billing = new CreditCardBilling();
            billing.chargeClient(client,150);

            then: "Client should have the bonus option active" #B
            client.hasBonus() == true
    }
    def "Client loses bonus if he does not accept the transaction"() {
            when: "a client buys something and later changes mind" #A
            def client = new Client();
            def billing = new CreditCardBilling();
            billing.chargeClient(client,150);
            client.rejectsCharge();

            then: "Client should have the bonus option inactive" #B
            client.hasBonus() == false
    }
}
```

#A Business description of test
#B human readable test result

Even if you are not a programmer, you can read just the English text in the code (sentences inside quotes) and get the following:

- Client should have a bonus if he spends more than 100 dollars
    - when a client buys something with value at least 100
    - then Client should have the bonus option active
- Client loses bonus if he does not accept the transaction
    - when a client buys something and later changes mind
    - then Client should have the bonus option inactive

This is very readable. A business analyst could read the test, and ask questions for other cases (what happens if the client spends 99.9? What happens if he changes his mind the next day and not immediately?)

Also if the second test breaks, the PM will see in the report a red bar with title "Client loses bonus if he does not accept the transaction." He instantly knows the severity of the problem (perhaps he decides to ship this version if he considers it non-critical)

Facts about Spock

- Spock is an alternative Test Framework written in the Groovy programming language
- A test framework automates the boring and repetitive process of manual testing which is essential for any large application codebase
- Although Spock is written in Groovy, it can test both Java and Groovy code
- Spock has built-in support for Mocking and Stubbing without an external library
- Spock follows the given-when-then code flow commonly associated with the Behavioral Driven Development paradigm
- Both Groovy and Java build and run on the JVM. A large enterprise build can run both JUnit and Spock tests in the same time.
- Spock uses the JUnit runner infrastructure and therefore is compatible with all existing Java infrastructure. For example, code coverage with Spock is possible in the same way as JUnit.
- One for the killer features of Spock is the detail it gives when a test fails. JUnit only mentions the expected and actual value, where Spock records the surrounding running environment mentioning the intermediate results and allowing the developer to pinpoint the problem with greater ease than JUnit
- Spock can pave the way for full Groovy migration into a Java project if that is what you wish. Otherwise it is perfectly possible to keep your existing JUnit tests in place and only use Spock in new code