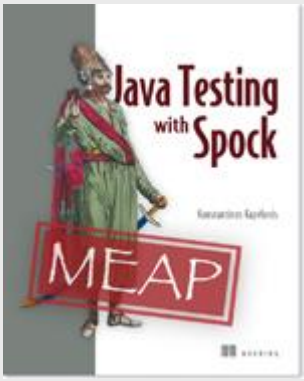

Spock: the groovier testing framework

By Konstantinos Kapelonis

If you always thought that JUnit (and/or TestNG) was the only solution for Java unit tests, guess again! Spock is an emerging test framework written in Groovy that can test both Java and Groovy code. It even includes its own Mocking mechanism. In this article adapted from Java testing with Spock, I will give you a high level overview of Spock and its capabilities.

When I first came upon Spock, I thought that it would be the jUnit alternative for the Groovy programming language. After all, once a programming language reaches a critical mass, somebody ports the standard testing model (known as xUnit) to the respective runtime environment. There are already xUnit frameworks for all popular programming languages.

But Spock is *not* the xUnit of Groovy! It resembles higher-level testing frameworks (such as Rspec and Cucumber) that follow the concepts of Behavioral-Driven development, instead of the basic setup-stimulate-assert style of xUnit. Behavioral Driven Development attempts (among other things) to create a one-to-one mapping between business requirements and unit tests.

Assert vs. Assertions

If you are familiar with JUnit, one of the first things you will notice with Spock is the complete lack of assert statements. Asserts are used in unit tests in order to verify the test. You define what is the expected result yourself and JUnit automatically fails the test if the expected output does not match the actual one. Assert statements are still there if you need them, but the preferred way is to use Spock "assertions" instead, a feature so powerful it has been in fact backported to Groovy itself.

AGNOSTIC TESTING OF JAVA AND GROOVY

Another unique advantage of Spock is the ability to test agnostically both Java and Groovy code as shown in figure 1.

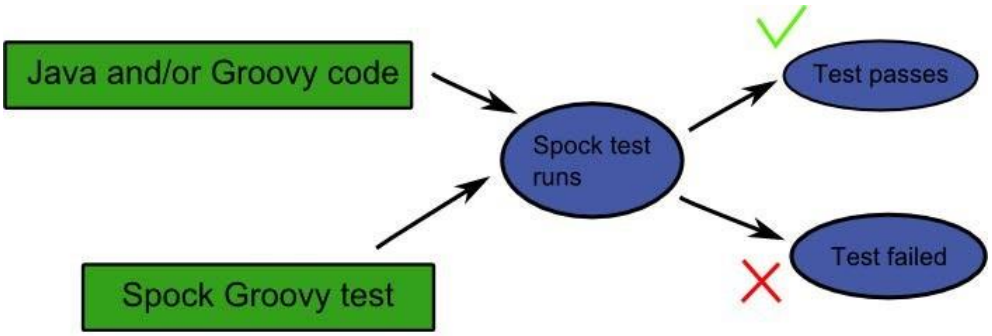

Figure 1. Spock can test both Java and Groovy code.

Groovy is a dynamic language that runs in the same Java Virtual Machine (JVM) as Java. Java supporters are very proud of the JVM and some believe that the value of the JVM as a runtime environment is even higher than Java the language. Spock is one example of the power the JVM has to accommodate code from different programming languages.

Spock can test any class that runs on the JVM, regardless of the original source code (Java or Groovy). It's possible with Spock to test either a Java class or a Groovy class in the exact same way. Spock doesn't care about the origin of the class, as long as it is JVM compatible. You can even verify both Java and Groovy code in the same Spock test if your project is a mix of the two

TAKING ADVANTAGE OF GROOVY TRICKS IN SPOCK TESTS

Finally, you need to know that Groovy is a dynamic language that behaves differently than Java in some important aspects (such as the declaration of variables. This means that several "tricks" you learn with Spock are in reality a mix of both Groovy and Spock magic, because Spock can extend Groovy syntax in ways that would be very difficult with Java (if not impossible). And yes, unlike Java, in Groovy it is possible for a library/framework to change the syntax of the code as well.

As you become more familiar with Spock and Groovy, the magic behind the curtain will start to appear and you might even be tempted to use Groovy outside of Spock tests as well!

AST transformations - changing the structure of Groovy language

Several tricks of Groovy magic come from the powerful meta-programming facilities offered during runtime that can change classes and methods in ways impossible with vanilla Java. At the same time Groovy also supports compile time macros (Abstract Syntax Tree transformations in Groovy parlance).

If you're already familiar with macros in other programming languages you should already be aware of the power they bring in code transformations. Using AST transformations a programmer

can add/change several syntactic features of Groovy code modifying the syntax in forms that were difficult/ impossible in Java.

Spock takes advantage of these compile and runtime code transformation features offered by Groovy in order to create a pseudo-DSL (domain specific language) specifically for unit tests.

Before starting with the details of Spock code, let's take a bird's eye view on its major features.

Enterprise testing

A test framework geared towards big enterprise application has certain requirements in order to handle the complexity and possible configurations that come with enterprise software. Such a test framework must easily adapt to the existing ecosystem of build tools, coverage metrics, quality dashboard and other automation facilities.

Rather than re-inventing the wheel, Spock bases its tests on the existing JUnit runner. The runner is responsible for executing JUnit tests and presenting their results to the console or other tools (e.g., the IDE). Spock reuses the JUnit runner to get for free all the mature support of external tools already created by JUnit.

- Do you want to see code coverage reports with Spock?
- Do you want to run your tests in parallel?
- Do you want to split your tests in long running and short running?

The answer to all these questions is "yes you can, as you did before with JUnit".

Data-driven tests

A common target for unit tests is to handle input data for the system in development. It is impossible to know all possible uses for your application in advance, let alone the ways people are going to use and misuse your application.

Usually a number of unit tests is dedicated to possible inputs of the system in a gradual way. The test starts with the known set of allowed or disallowed input, and as bugs are encountered, it is enriched with more cases. Common examples would be a test that checks if a username is valid or not, or what date formats are accepted in a web service.

These tests suffer from a lot of code duplication if code is handled carelessly. The test is always the same (is the username valid or not?) and only the input changes. While JUnit has some facilities for this type of tests (parameterized tests), Spock takes a different turn allowing you to actually embed data tables in Groovy source code. Spock offers a special DSL (Domain Specific Language) that allows tabular data in the unit test source code.

Mocking and Stubbing

For all its strengths, Object Oriented Software suffers from an important flow. The fact that two objects work correctly individually does not imply that both objects will also work correctly when connected to each other. The reverse is also true, side effects from an object chain may hide or mask problems that happen in individual class.

A direct result from this flow is that testing OOP software usually needs to cover two levels at once. The integration level where tests examine the system as a whole (integration tests) and the class level where tests examine each individual class (unit tests or logic tests).

In order to examine the microscopic level of a single class and isolate it out of the macroscopic level of the system, a controlled running environment is needed. A developer has to focus on a single class and the rest of the system is to be assumed "correct." Attempting to test a single class inside the real system is very difficult because any bugs encountered are not immediately clear if they happen because of the class under test or the environment.

To this purpose, a mocking framework is needed that "fakes" the rest of the system and leaves only the class under test to be "real." The class is then tested in isolation because even though it "thinks" that it is inside a real system, in reality, all other collaborating classes (collaborators) are simple puppets with pre-programmed input and output.

In the JUnit world an external library is needed for mocking. Therse are numerous with both strengths and weaknesses at the same time (Mockito, jMock, Easymock, Powermock). Spock comes with its own mocking framework built-in. Combined with the power of Groovy metaprogramming, Spock is a comprehensive Domain Specific Language that provides all puzzle pieces needed for testing.