Spock

the enterprise ready specification framework

Sameer Garg

Spock

the enterprise ready specification framework

Sameer Garg

What Spock is ?

✦Specification + Mock
✦Groovy DSL for Testing
✦Builds on JUnit 4
✦Goals: Concise, Maintainable
✦Given / When / Then
✦Authors:

- Peter Niederwieser
- Luke Daley

Reasons to use Spock over JUnit

- A unit test *is* documentation, so readability matters. Spock tests make intuitive sense at first glance.
- Less lines of code
- Writing tests is an integral part of software development. If you can write them more quickly, you win lots of time. Spock's DSL for testing allows precisely that.
- Can be extended
- Turn tests into specifications
- Low learning curve

Introduction to groovy

Introduction to groovy

```
//declarations
def color=red
def personMap = [:]
def coloursList = []
//populate lists
coloursList<<[“red”,”blue”]
//groovy truth
“red”==colour || colourLists
//groovy string
def statement = “my car is of $color  color”
//elvis operator
user?.street?.address
```

Spock Users

Gradle GPars

Grails Geb

Grails Plugin Collective

Apache Tapestry

Griffon Spock

Spring?

Betfair be2

bemoko BSkyB CTI Digital

Donewtech Solutions

eHarmony

Energized Work Gennemtænkt IT

IntelliGrape Software Netflix

Smarter Ecommerce STERMEDIA Sp. z o.o.

Software Projects Spot Diving

SystemsForge TouK

Fixture methods

```
def setup() {}
    run before every feature method
def cleanup() {}
    run after every feature method
def setupSpec() {}
    run before first feature method
def cleanupSpec() {}
    run after last feature method
```

Like JUnit 4:

```
    @Before, @After,
    @BeforeClass, @AfterClass
```

Block

setup: cleanup:

given:

when:

Stimulus

then:

Response, booleans are checked

expect: where:

An `expect` block is more limited than a `then` block in that it may only contain conditions and variable definitions. It is useful in situations where it is more natural to describe stimulus and expected response in a single expression

First Specification

SkySpecification.groovy

```
package org.example

import spock.lang.*

class SkySpecification extends Specification {

  ...
}
```

All Specifications extend from this base class

Sky.java

```
package org.example.sus;

public class Sky {
  public String getColor() {
    return "blue";
  }
}
```

Spock in Action

State Based Testing

- **Classical Unit Testing**

  Arrange
  Act
  Assert

OR

- **Given-When-Then**

State based testing

- **Blocks**

  setup: cleanup: expect: given: when: then:
  where: and:

- **Fixture Methods**
  - setup() cleanup() setupSpec() cleanupSpec()

- **Instance and** @Shared **fields**

- old() **and** thrown()

Account.java

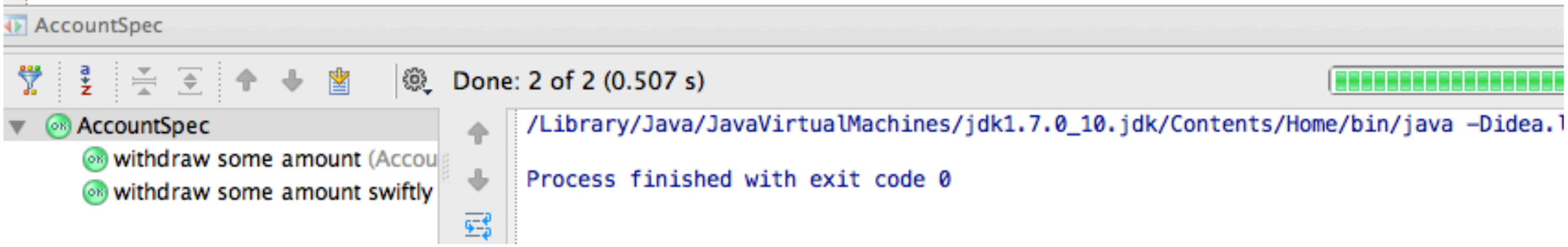
```
public class Account {
    private BigDecimal balance = new BigDecimal(0);

    public Account(BigDecimal initial) {
        balance = initial;
    }

    public BigDecimal getBalance() {
        return balance;
    }

    public void withdraw(BigDecimal amount) {
        if  (amount.compareTo(BigDecimal.ZERO) < 0) {
            throw new NegativeAmountWithdrawnException(amount);
        }
        balance = balance.subtract(amount);
    }
```

NegativeAmountWithdrawnException.java

```
public class NegativeAmountWithdrawnException extends RuntimeException {
    private final BigDecimal amount;

    public NegativeAmountWithdrawnException(BigDecimal amount) {
        super("cannot withdraw" + amount);
        this.amount = amount;
    }

    public BigDecimal getAmount() {
        return amount;
    }
}
```

```groovy
import spock.lang.Specification

class AccountSpec extends Specification {

    def "withdraw some amount"() {
        given:
        Account account = new Account(BigDecimal.valueOf(5));

        when:
        account.withdraw(BigDecimal.valueOf(2));

        then:
        account.getBalance() == BigDecimal.valueOf(3);
    }

    def "withdraw some amount swiftly"() {
        given:
        def account = new Account(5.0)

        when:
        account.withdraw(2.0)

        then:
        account.balance == 3.0
    }
}
```

More details inside code



Data driven tests

- where: **block**
- Data tables
- External data sources
- @Unroll

Interaction based testing

- **Creating**

```
def sub = Mock(Subscriber)
Subscriber sub = Mock()
```

- **Mocking**

```
1 * sub.receive("msg")
(1..3) * sub.receive(_)
(1.._) * sub.receive(_ as String)
1 * sub.receive(!null)
1 * sub.receive({it.contains("m")})
1 * _./rec.*/("msg")
```

Stubbing

```
// now returns status code
String receive(String msg) { ... }
// setting expectation
sub.receive(_) >> "ok"
sub.receive(_) >>> ["ok", "ok", "fail"]
sub.receive(_) >>> { msg -> msg.size() > 3 ? "ok" : "fail" }
```

Mocking and Stubbing

```
3 * sub.receive(_) >>> ["ok", "ok", "fail"]
// Any set method with one arg
pojo./set.*/(_)
```

Built in extensions

- **@Ignore:** Ignores a feature method.

- **@IgnoreRest:** Ignores all feature methods not carrying this annotation.
  Useful for quickly running just a single method.
- **@FailsWith:** Expects a feature method to complete abruptly.
  @FailsWith has two use cases: First, to document known bugs that cannot be resolved immediately. Second, to replace exception conditions in certain corner cases where the latter cannot be used (like specifying the behavior of exception conditions). In all other cases, exception conditions are preferable.
- **@Timeout:** Sets a timeout for execution of a feature or fixture method.
- **@AutoCleanup:** Automatically cleans up the object stored in the annotated field or property
- **@Stepwise:** Runs feature method in order of declaration
- **@RevertMetaClass:** Revert the meta class of the given classes

Resources

- **Homepage**

  http://spockframework.org

- **Source Code**

  https://github.com/spockframework/spock

- **Spock Web Console**

  http://meet.spockframework.org

- **Spock Example Project**

  http://downloads.spockframework.org
  https://github.com/spockframework/spock/tree/groovy-1.8/spock-example