

Spock Framework

Fernando Camargo

2 de junho de 2017

ZG Soluções

Spock?



- Framework de testes

Spock Framework

- Framework de testes
- Para aplicações Java e Groovy

Spock Framework

- Framework de testes
- Para aplicações Java e Groovy
- Baseado em Groovy

Spock Framework

- Framework de testes
- Para aplicações Java e Groovy
- Baseado em Groovy
- Compatível com JUnit

- Menor código de testes

- Menor código de testes
- Legibilidade de testes

- Menor código de testes
- Legibilidade de testes
- Testes → Especificações

- Menor código de testes
- Legibilidade de testes
- Testes → Especificações
- Extensível

- Menor código de testes
- Legibilidade de testes
- Testes → Especificações
- Extensível
- Especificações como documentação

State Based Testing

State Based Testing

- Testes que invocam métodos e validam o estado do objeto sob testes

- Testes que invocam métodos e validam o estado do objeto sob testes
- Testes clássicos:
 - Arranjar
 - Agir
 - Verificar (Assert)

State Based Testing

- Testes que invocam métodos e validam o estado do objeto sob testes
- Testes clássicos:
 - Arranjar
 - Agir
 - Verificar (Assert)
- Given-When-Then

Teste de Account com JUnit Clássico

```
public class AccountTest {  
    @Test  
    public void withdrawSomeAmount(){  
        // given  
        Account account = new Account(BigDecimal.valueOf(5));  
  
        // when  
        account.withdraw(BigDecimal.valueOf(2));  
  
        // then  
        assertEquals(BigDecimal.valueOf(3), account.getBalance());  
    }  
}
```

Teste de Account com Spock (1)

```
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)  
    }  
  
}
```

Teste de Account com Spock (2)

```
class AccountSpec extends Specification {  
  
    def "withdraw some amount"(){  
        given:  
            Account account = new Account(5.0)  
  
        when:  
            account.withdraw(2.0)  
  
        then:  
            account.balance == 3.0  
    }  
  
}
```

Teste de Account com Spock (3)

```
class AccountSpec extends Specification {  
  
    def "withdraw some amount"(){  
        given: "an account with a balance of five euros"  
        Account account = new Account(5.0)  
  
        when: "two euros are withdrawn"  
        account.withdraw(2.0)  
  
        then: "three euros remain in the account"  
        account.balance == 3.0  
    }  
  
}
```

Teste de Account com Spock (4)

```
class AccountSpec extends Specification {

    def "withdrawing some amount decreases the balance by exactly that amount"(){
        given: "an account with a balance of five euros"
        Account account = new Account(5.0)

        when: "two euros are withdrawn"
        account.withdraw(2.0)

        then: "three euros remain in the account"
        account.balance == 3.0
    }

    def "can't withdraw a negative amount"(){
        given: "an account with a balance of five euros"
        Account account = new Account(5.0)

        when: "trying to withdraw -1"
        account.withdraw(-1.0)

        then: "an exception is thrown"
        NegativeAmountWithdrawnException e = thrown()
        e.amount == -1.0
    }
}
```

Conceitos básicos

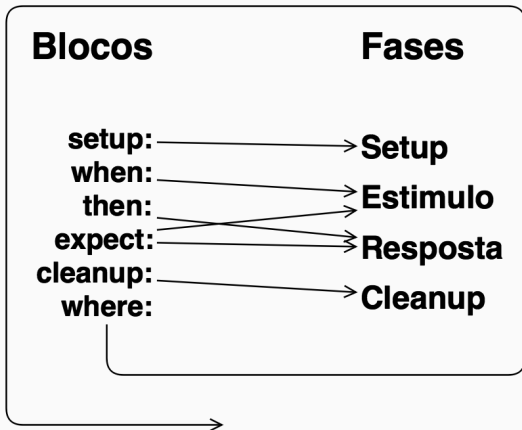
Specification

```
import spock.lang.*

class MyFirstSpecification extends Specification {
    // fields
    def col = new Collaborator()
    @Shared
    def sharedCol = new Collaborator()

    // fixture methods
    def setup() {}
    def cleanup() {}
    def setupSpec() {}
    def cleanupSpec() {}

    // feature methods
    // helper methods
}
```



- Primeiro bloco de um método de testes

- Primeiro bloco de um método de testes
- Faz as preparações necessárias para o teste

- Primeiro bloco de um método de testes
- Faz as preparações necessárias para o teste
- Pode começar com `setup:`, `given:` ou implícito

Blocos de Estímulo e Resposta

- **when:**
 - Descreve um estímulo (invocação de método(s))

- **when:**
 - Descreve um estímulo (invocação de método(s))
 - Pode conter código arbitrário

Blocos de Estímulo e Resposta

- **when:**
 - Descreve um estímulo (invocação de método(s))
 - Pode conter código arbitrário
- **then:**
 - Descreve uma resposta

Blocos de Estímulo e Resposta

- **when:**
 - Descreve um estímulo (invocação de método(s))
 - Pode conter código arbitrário
- **then:**
 - Descreve uma resposta
 - Restrito a condições, interações e definições de variável

Blocos de Estímulo e Resposta

- **when:**
 - Descreve um estímulo (invocação de método(s))
 - Pode conter código arbitrário
- **then:**
 - Descreve uma resposta
 - Restrito a condições, interações e definições de variável
- **expect:**
 - Descreve estímulo e resposta numa única expressão

Blocos de Estímulo e Resposta

- **when:**
 - Descreve um estímulo (invocação de método(s))
 - Pode conter código arbitrário
- **then:**
 - Descreve uma resposta
 - Restrito a condições, interações e definições de variável
- **expect:**
 - Descreve estímulo e resposta numa única expressão

when-then → métodos com efeitos colaterais/alteram estado

expect → métodos funcionais

Blocos de Estímulo e Resposta

```
import spock.lang.*

class StimulusResponseSpecification extends Specification {

    def "when-then style"(){
        when:
        int x = Math.max(5, 9)

        then:
        x == 9
    }

    def "expect style"(){
        expect:
        Math.max(5, 9) == 9
    }
}
```

- Último bloco de um método de testes (podendo ser seguido apenas por where)

- Último bloco de um método de testes (podendo ser seguido apenas por where)
- Limpeza de recursos externos

- Último bloco de um método de testes (podendo ser seguido apenas por where)
- Limpeza de recursos externos
- Sempre começa com **cleanup:**

Data Driven Testing

- Testa o mesmo comportamento com dados variáveis

- Testa o mesmo comportamento com dados variáveis
- O mesmo teste é executado múltiplas vezes

- Testa o mesmo comportamento com dados variáveis
- O mesmo teste é executado múltiplas vezes
- @Unroll faz reportar como testes diferentes

Teste de Account com Spock (5)

```
class AccountSpec extends Specification {  
  
    def "withdrawing some amount decreases the balance by exactly that amount"(){  
        given: "an account with a balance of five euros"  
        Account account = new Account(balance)  
  
        when: "two euros are withdrawn"  
        account.withdraw(withdrawn)  
  
        then: "three euros remain in the account"  
        account.balance == remaining  
  
        where:  
        balance | withdrawn || remaining  
        5.0      | 2.0       || 3.0  
        4.0      | 0.0       || 4.0  
        4.0      | 4.0       || 0.0  
    }  
  
}
```

Interaction Based Testing

- Comunicação entre objetos (invocação de métodos)

- Comunicação entre objetos (invocação de métodos)
- Spock possui framework próprio de Mocking

Teste de Publisher/Subscriber com Spock (1)

```
class PublisherSubscriberSpec extends Specification {

    Publisher pub = new Publisher()
    Subscriber sub1 = Mock()
    Subscriber sub2 = Mock()

    def "deliver messages to all subscribers"(){
        when:
            pub.publish("msg")

        then:
            1 * sub1.receive("msg")
            1 * sub2.receive("msg")
    }
}
```

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

Mocking/Stubbing com Spock

- Criação
 - `def sub = Mock(Subscriber)`
 - `Subscriber sub = Mock()`
- Mocking (verificação)
 - `1 * sub.receive("msg")`

Mocking/Stubbing com Spock

- Criação
 - `def sub = Mock(Subscriber)`
 - `Subscriber sub = Mock()`
- Mocking (verificação)
 - `1 * sub.receive("msg")`
 - `(1..3) * sub.receive("msg")`

Mocking/Stubbing com Spock

- Criação

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

- Mocking (verificação)

- `1 * sub.receive("msg")`
- `(1..3) * sub.receive("msg")`
- `(1.._) * sub.receive(_ as String)`

- Criação

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

- Mocking (verificação)

- `1 * sub.receive("msg")`
- `(1..3) * sub.receive("msg")`
- `(1.._) * sub.receive(_ as String)`
- `1 * sub.receive(!null)`

- Criação

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

- Mocking (verificação)

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

- Criação

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

- Mocking (verificação)

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

- Stubbing (sem verificação)
 - `sub.receive(_) >> "ok"`

- Stubbing (sem verificação)
 - `sub.receive(_) >> "ok"`
 - `sub.receive(_) >>> ["ok", "ok", "fail"]`

- Stubbing (sem verificação)
 - `sub.receive(_) >> "ok"`
 - `sub.receive(_) >>> ["ok", "ok", "fail"]`
 - `sub.receive(_) >>> {msg -> msg ? "ok" : "fail"}`

- Stubbing (sem verificação)
 - `sub.receive(_) >> "ok"`
 - `sub.receive(_) >>> ["ok", "ok", "fail"]`
 - `sub.receive(_) >>> {msg -> msg ? "ok" : "fail"}`
- Mocking + Stubbing
 - `3 * sub.receive(_) >>> ["ok", "ok", "fail"]`

Conclusões

Spock vs JUnit

Spock	JUnit
Specification	Test class
setup()	@Before
cleanup()	@After
setupSpec()	@BeforeClass
cleanupSpec()	@AfterClass
Feature	Test
Feature method	Test method
Data-driven feature	Theory
Condition	Assertion
Exception condition	@Test(expected=...)
Interaction	Mock expectation

- Código mais conciso

- Código mais conciso
- Funcionalidades do Groovy para os testes

- Código mais conciso
- Funcionalidades do Groovy para os testes
- Documentação de especificações