

Spock - The Logical Enterprise Testing Tool

Ken Sipe





<http://kensipe.blogspot.com/>

<http://del.icio.us/kensipe>

twitter: @kensipe

ken.sipe@gmail.com

Developer: Embedded, C++, Java, Groovy, Grails, C#, Objective C

Speaker: JavaOne 2009 Rock Star, NFJS, JAX

Microsoft MCP

Sun Certified Java 2 Architect

Master of Scrums

Agile Coach

Instructor: VisiBroker CORBA

Rational Rose, OOAD





- Testing
- Introduction to Spock
- Mocking



Spock Intro



- testing framework...
- based on Groovy
- fully compatible with JUnit
- result of learnings from
 - RSpec, BDD, JUnit



- Reduces lines of code
- Make tests more readable
- Be extended

JUnit

```
public class SimpleInterestCalculatorJUnitTest extends TestCase {  
  
    InterestCalculator interestCalculator;  
  
    protected void setUp() throws Exception {  
        interestCalculator = new SimpleInterestCalculator();  
        interestCalculator.setRate(0.05);  
    }  
  
    public void testCalculate() {  
        double interest = interestCalculator.calculate(10000, 2);  
        assertEquals(interest, 1000.0);  
    }  
  
    public void testIllegalCalculate() {  
        try {  
            interestCalculator.calculate(-10000, 2);  
            fail("No exception on illegal argument");  
        } catch (IllegalArgumentException e) {  
        }  
    }  
  
}
```

JUnit Pain Points?

- Concise, Clear and Readable
- Promote “user” thinking
 - context
 - stimulus
 - expectations
- Productivity

Groovy

```
class SimpleInterestCalculatorGTest extends GroovyTestCase {  
  
    def interestCalculator;  
  
    protected void setUp() throws Exception {  
        interestCalculator = new SimpleInterestCalculator(rate: 0.05)  
    }  
  
    public void testCalculate() {  
        double interest = interestCalculator.calculate(10000, 2)  
        assertEquals interest, 1000.0  
    }  
  
    public void testIllegalCalculate() {  
        shouldFail {  
            interestCalculator.calculate(-10000, 2)  
        }  
    }  
}
```

```
class gInMemoryAccountDaoTests extends GroovyTestCase {
    def EXISTING_ACCOUNT_NO = "1234"
    def NEW_ACCOUNT_NO = "5678"

    Account existingAccount
    Account newAccount
    InMemoryAccountDao accountDao

}

protected void setUp() {
    super.setUp()
    existingAccount = new Account(EXISTING_ACCOUNT_NO, 100)
    newAccount = new Account(NEW_ACCOUNT_NO, 200)
    accountDao = new InMemoryAccountDao()
    accountDao.createAccount(existingAccount)
}

}

public void testAccountExists() {
    assert accountDao.accountExists(EXISTING_ACCOUNT_NO)
    assert (accountDao.accountExists(NEW_ACCOUNT_NO)) == false
}

}

public void testCreateNewAccount() {
    accountDao.createAccount(newAccount)
    assert accountDao.findAccount(NEW_ACCOUNT_NO) == newAccount
}

}

public void findNotExistedAccount() {
    shouldFail {
        accountDao.findAccount(NEW_ACCOUNT_NO)
    }
}

}
```

Condition not satisfied:

```
interest == calc.calculate(amt, year)
```

```
  |      | |           |      | |  
25.0  | |           20.0 100 4
```

```
      | com.math.SimpleInterestCalculator@606e1dec  
false
```

- Concise, Clear and Readable
- Promote “user” thinking
 - context
 - stimulus
 - expectations
- Productivity

Spock

- Programmers Environment
 - Groovy
- Promotes Clarity
 - structural blocks
 - removes noise



- Expressive testing language
- Easy to learn
- Usable from unit to end-to-end
- Leverages Groovy
- Runs with JUnit Runner
 - IDE
 - CI

Taxonomy of a Spec

```
import spock.lang.Specification

class MyFirstSpec extends Specification {

    //fields
    //fixture methods
    // feature methods
    // helper methods
}
```



■ Specification

- compare to TestCase or GroovyTestCase
- Instructs JUnit to run with **Sputnik** (JUnit runner)

■ Fields

- initialized for each “test”
- think “setup”
- not shared between feature methods



```
@Shared res = new VeryExpensiveResource()
```

■ Shared

- Setup once
- think setupSpec()

■ statics

- only use for constants



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

- before / after a feature
- before / after a spec
- optional



```
// feature methods
def "pushing an element on the stack"() {
    // blocks go here
}
```

- “heart” of spec
- four phases
 - setup the features fixture
 - provide stimulus to system
 - describes the response
 - clean up

given:	preconditions, data fixtures
when:	actions that trigger some outcome
then:	makes assertions about outcome
expect:	short alt to when & then
where:	applies varied inputs
and:	sub-divides other blocks
setup:	alias for given
cleanup:	post-conditions, housekeeping



- When / Then / Where
- Given / When / Then

■ setup

- must be first
- must be the only
- no special semantics
- label is optional
- label given: is an alias

```
setup:  
def stack = new Stack()  
def elem = "push me"
```

```
given: "setup and initialization of ..."  
def stack = new Stack()  
def elem = "push me"
```



```
when: // stimulus
stack.push(elem)

then: // response
!stack.empty
stack.size() == 1
stack.peek() == elem
```

- used together
 - possible to have many per feature
- then restrictions
 - conditions
 - exception conditions
 - automatic asserts
 - interactions
 - variable defs



```
when:  
stack.pop()
```

```
then:  
thrown(EmptyStackException)  
stack.empty
```

```
when:  
stack.pop()
```

```
then:  
EmptyStackException e = thrown()  
e.cause == null
```

■ checking for exceptions



```
def "HashMap accepts null key"() {  
  setup:  
    def map = new HashMap()  
  
  when:  
    map.put(null, "elem")  
  
  then:  
    notThrown(NullPointerException)  
}
```



```
def "events are published to all subscribers"() {  
  def subscriber1 = Mock(Subscriber)  
  def subscriber2 = Mock(Subscriber)  
  def publisher = new Publisher()  
  publisher.add(subscriber1)  
  publisher.add(subscriber2)  
  
  when:  
    publisher.fire("event")  
  
  then:  
    1 * subscriber1.receive("event")  
    1 * subscriber2.receive("event")  
}
```



```
def "setup and cleanup example"() {  
  setup:  
  def file = new File("/some/path")  
  file.createNewFile()  
  
  // ...  
  
  cleanup:  
  file.delete()  
}
```

- cleanup block
 - only followed by a where block
 - no repeats



```
def "computing the maximum of two numbers"() {  
  expect:  
  Math.max(a, b) == c  
  
  where:  
  a << [5, 3]  
  b << [1, 9]  
  c << [5, 9]  
}
```

- last in a method
- no repeats
- used for data-driven features



```
def "offered PC matches preferred configuration"() {
  when:
  def pc = shop.buyPc()

  then:
  matchesPreferredConfiguration(pc)
}

// helper methods
def matchesPreferredConfiguration(pc) {
  pc.vendor == "Sunny" && pc.clockRate >= 2333
}

void matchesPreferredConfiguration(pc) {
  assert pc.vendor == "Sunny"
  assert pc.clockRate >= 2333
}
```

- either return a boolean
 - or
- assert

Specification Functions

and Spock.lang.*

- `old()`
- `thrown()` / `notThrown`
- `with {}`

@Title
@Narrative
@Issue
@See
@Subject



@Requires
@IgnoreSelf
@Ignore
@IgnoreRest



@Unroll



`@AutoCleanup`

`@AutoCleanup('dispose')`

`@AutoCleanup(quite=true)`



@Timeout

@Timeout(10)

@Timeout(value=10,
unit=TimeUnit.MILLISECONDS)



@Stepwise

Hamcrest

JUnit Rules

SpockConfig



■ Getting Spock

- <http://code.google.com/p/spock/>

■ Source from Presentation

- <https://github.com/kensipe/spock-demos-nfjs>

■ Closing and Q&A

- Please fill out the session evaluation
- Ken Sipe
 - ken.sipe@gmail.com
 - kensipe.blogspot.com
 - twitter: @kensipe