JavaOne 2014

Advanced Java Unit Testing

with Spock

@kensipe

NFJS
No Fluff Just Stuff.com

About Speaker

http://kensipe.blogspot.com/
http://del.icio.us/kensipe
twitter: @kensipe
ken.sipe@gmail.com

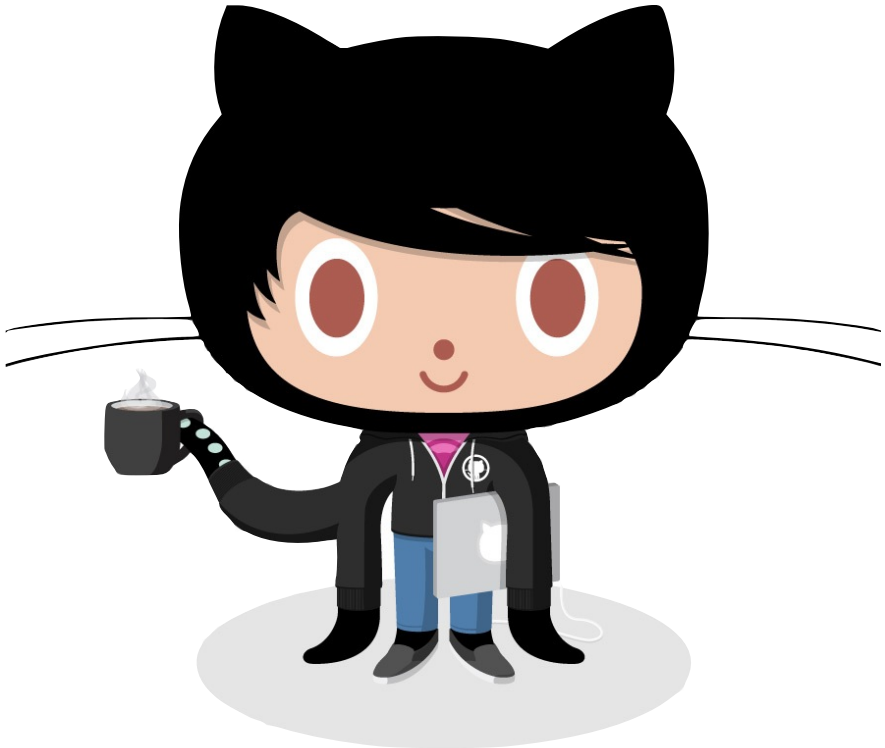
Developer: Embedded, C++, Java, Groovy, Grails, C#, Objective C
Speaker: JavaOne 2009 Rock Star, NFJS, JAX
Microsoft MCP
Sun Certified Java 2 Architect
Master of Scrums
Agile Coach
Instructor: VisiBroker CORBA
Rational Rose, OOAD

Agenda

- Testing
- Spock Basics
- Spock Mocking
- Advanced Spock

https://github.com/kensipe/spock-

Code Intensive



Spock Intro

Spock is...

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

Spock Can...

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

JUnit

Taxonomy of JUnit

```java
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?

Verbose!!!

There was a time when I only survived because I made myself write out three pages of words in the morning. Pen to paper: no coming up for air until there were three full pages, and don't you dare read them, no. No re-reading until at least a month has gone by. The whole idea came from Julie Cameron and her Artist's Way book. I never really got past the first or second chapter, but they sure served me well. Well that survival may have been mostly just an emotional one, but still, I survived. I was the wreck of the Titanic. My grandma taught me that phrase. I should call her. Some days I pull out those old survival pages and re-read them. I laugh at myself. I say, "hmmmmm." I say, "I'm glad I made myself articulate those days." And then I stuff them back into the old manila envelope where I stack them, three pages by three. And I wonder, where were those words before I wrote them all down? Maybe they were crouched behind wells of tears, or hiding behind walls of jealousy. Maybe they were shaking hands with old memories, and trying to teach them some rather unimpressive breakdance moves. I'm not certain. But it always felt better to write them down. Some days it had the effect of standing on the edge of an oceanic cliff and screaming for ten minutes. Other days I could hardly see through my tears, and snotted all over the page. Some days I had to write, "blah blah blah I don't know what to write about this morning" for six lines before I realized I had six pages of things to dislodge in my mind. I have friends who don't care for words. And I have friends who are enamored by them. I have friends who treat them cheaply. I have friends who don't know how well they play with them. And I have friends who have so many words locked up in them. I think that there is a lot more most of us need to say. But we're not sure if there's enough time to let the children out to play. We're not sure if our neighbor is safe enough to watch them, yet. Maybe tomorrow, we say. I think a wise old man once said that there is time for words and there is a time for silence, or something like that. A time for war and a time and for peace, a time to embrace and a time to refrain. I think that's the one. I like that in the beginning, there was the Word. I think I underestimate how much of a wordsmith God really is.

Wordy

Repetitive and Repetitive

Refactoring Nightmare

Groovy

Taxonomy of GroovyTestCase

```
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)
        }
    }

}
```

Groovy Power Assert

```
Condition not satisfied:

interest == calc.calculate(amt, year)
     |       | |               |        |    |
   25.0      | |              20.0 100  4
             |  com.math.SimpleInterestCalculator@606e1dec
           false
```

Groovy Improvements

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

Spock

Spock

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

Why use Spock?

- 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

First Spec


```
import spock.lang.Specification

class MyFirstSpec extends Specification {

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

Terms

- 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

More Fields

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

- Shared
  - Setup once
  - think setupSpec()
- statics
  - only use for constants

Fixture Methods

```
//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

```
// 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

Blocks

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

Blocks

- setup

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

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

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

When / Then Blocks

```
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

Conditions

```
when:
stack.pop()

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

```
when:
stack.pop()

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

- checking for exceptions

Throwing an Exception NOT!


```
def "HashMap accepts null key"() {
    setup:
    def map = new HashMap()

    when:
    map.put(null, "elem")

    then:
    notThrown(NullPointerException)
}
```

Mocks and Expectations

```
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")
}
```

Cleanup on Feature X

```
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

Where Blocks

```
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

Helper Methods

```
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

Demo Basics

power assert (1)
simple calc (1)
data driven with unroll (1)(7)(8)
hamcrest (1)

Specification Functions

and Spock.lang.*

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

Spock.lang.* Documentation

@Title
@Narrative
@Issue
@See
@Subject

Spock.lang.* Ignoring Stuff

@Requires
@IgnoreIf
@Ignore
@IgnoreRest

@Unroll

Built-in Extensions: AutoCleanup

@AutoCleanup
@AutoCleanup(‘dispose’)
@AutoCleanup(quite=true)

Built-in Extensions: Timeout

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

Demo Basics

old (2)
with (3)
requires java8 (4)
fast slow (5)

If they can handle it…
TableOfClosures

The need to Fake it

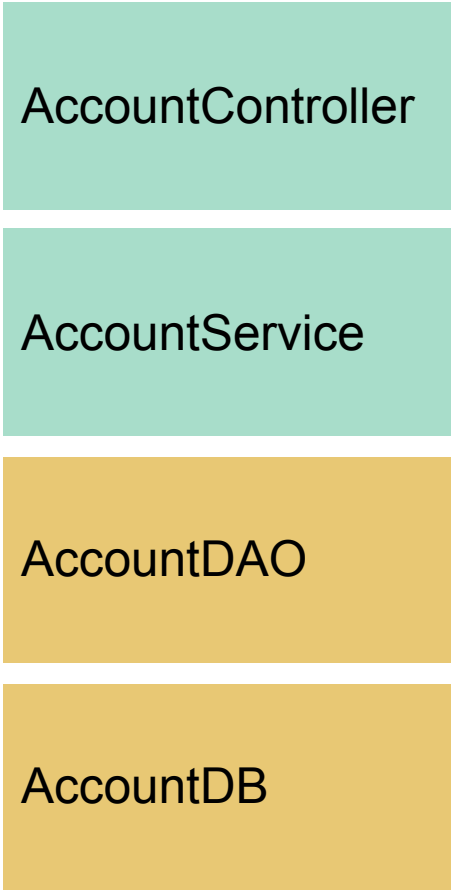
Why Do We Fake it?

Reduce the setup overhead

Test Isolation

- Focus on a specific concern

Increase testing performance



Mocks Aren't Stubs

Mocks Aren't Stubs

martinfowler.com/articles/mocksArentStubs.html

MARTIN FOWLER

Intro Design Agile Refactoring NoSQL DSL Delivery About Me ThoughtWorks

Mocks Aren't Stubs

The term 'Mock Objects' has become a popular one to describe special case objects that mimic real objects for testing. Most language environments now have frameworks that make it easy to create mock objects. What's often not realized, however, is that mock objects are but one form of special case test object, one that enables a different style of testing. In this article I'll explain how mock objects work, how they encourage testing based on behavior verification, and how the community around them uses them to develop a different style of testing.

02 January 2007

Martin Fowler

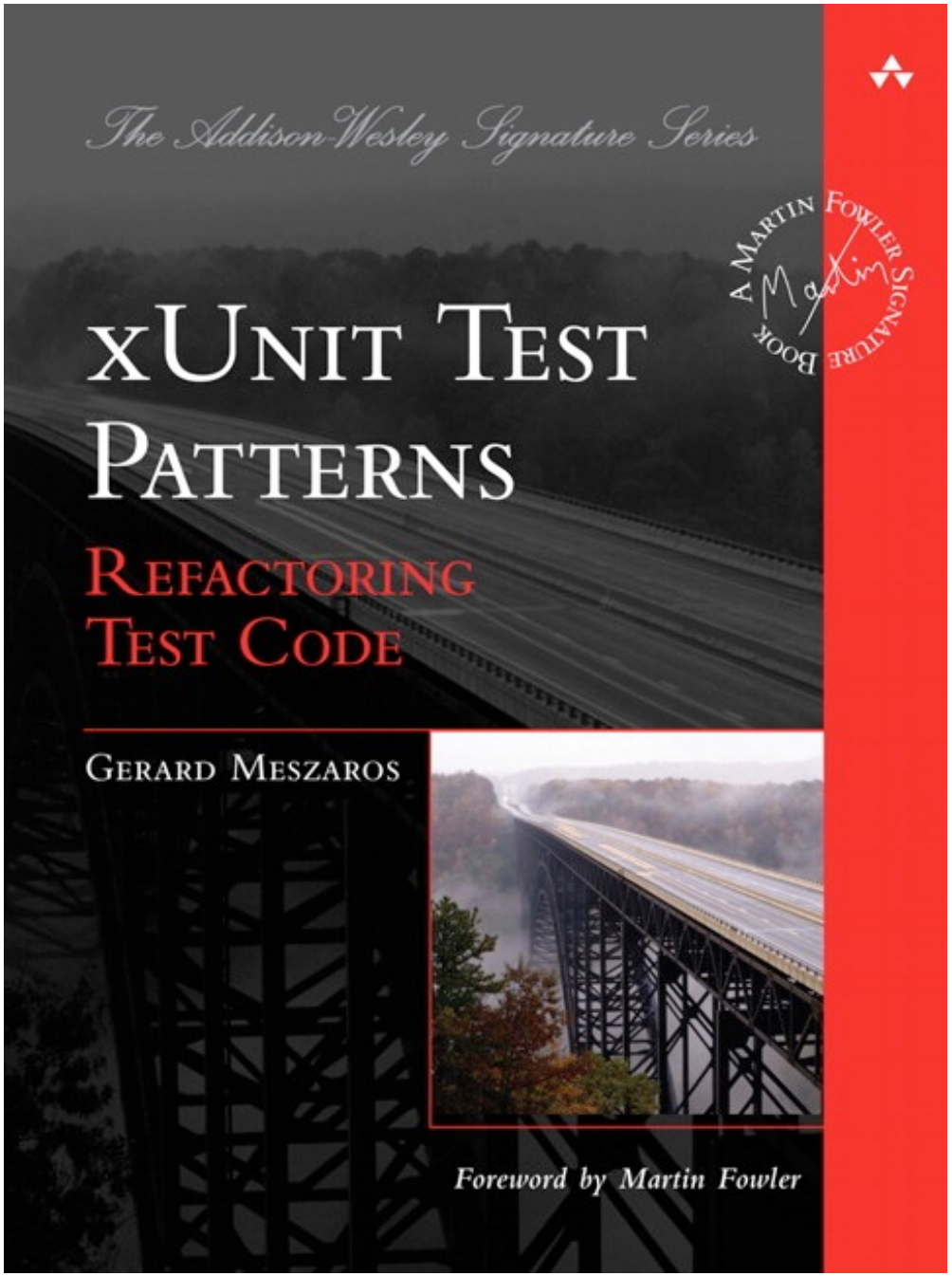
Translations: French Italian Spanish Portuguese

Tags: popular · testing

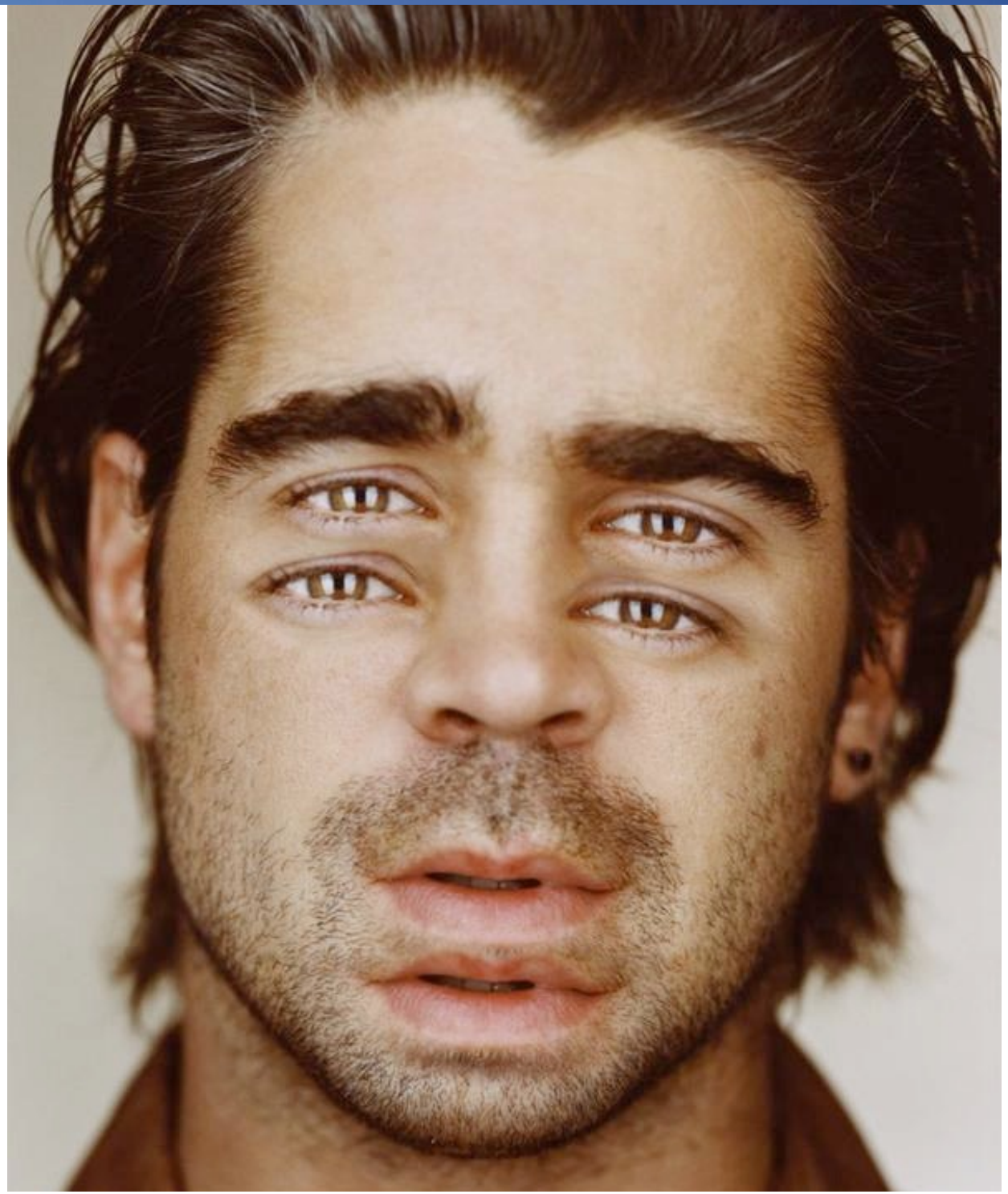
Contents

I first came across the term "mock object" a few years ago in the XP community. Since then I've run into mock objects more and more. Partly this is because many of the leading developers of mock objects have been colleagues of mine at ThoughtWorks at various times. Partly it's because I see them more and more in the XP-influenced testing

Mocks Aren't Stubs



Test Double



Dummy

- Dummy
  - Objects passed around
    - you don't care about them
    - but they are required (parameters lists, etc.)

Fake

- Fake
  - Working implementations with built-in short-cuts
    - not production
    - in-memory database

Fake

- Fake
  - Working implementations with built-in short-cuts
    - not production
    - in-memory database

Stub

- Stub
  - Objects with “canned” answers
  - Some times record interactions

Mocks

- Mock
  - Objects with **pre-programmed** expectations
  - Testing for behavior instead of state

Solutions

Type	Tool Options
Dummy	- developer only - objenesis
Fake	- db options - hibernate
Stubs	
Mocks	

- Classical TDD
- Mockist TDD
- Behavior Driven Development
  - off-shoot of mockist

Faking it with Spock

Spock Mock

- Spock Mock
  - Response with:
    - zero
    - null
  - is **verified**

```
/**
 * A mock object whose method calls are verified, which instantiates class-based mock objects with Objenesis,
 * and whose strategy for responding to unexpected method calls is {@link ZeroOrNullResponse}.
 */
MOCK(true, true, ZeroOrNullResponse.INSTANCE),
```

Spock Mock

- Spock Stub
  - Response with:
    - Empty
    - Dummy object
  - is **not** verified

```
/**
 * A mock object whose method calls are not verified, which instantiates class-based mock objects with Objenesis,
 * and whose strategy for responding to unexpected method calls is {@link EmptyOrDummyResponse}.
 */
STUB(false, true, EmptyOrDummyResponse.INSTANCE),
```

Spock Mock

- Spock Spy
  - Response with:
    - Delegated calls to real object
    - unless programmed to do otherwise
  - is **verified**

```
/**
 * A mock object whose method calls are verified, which instantiates class-based mock objects by calling a
 * real constructor, and whose strategy for responding to unexpected method calls is {@link CallRealMethodResponse}.
 */
SPY(true, false, CallRealMethodResponse.INSTANCE);
```

JMock vs. EasyMock vs. Spock

Mocking

Publisher
Spying
JUnit Rules

Mocking with Spock

```
def catalogService = Mock(CatalogService)

CatalogService catalogService = Mock()
```

```
void "test interaction scoping"() {
    given:"create mock CatalogService"
        CatalogService service = Mock()
        controller.catalogService = service
    and: "make sure it can file books"
        service.isAvailable(_ as Book) >> true

    when:"we verify isdn"
        params.isdn = book.isdn
        def result = controller.verifyISDN()

    then:"verify the book was filed"
        1 * service.inquired(_)
        result == true
}
```

```
void "test interaction scoping"() {
    given:"create mock CatalogService"
        CatalogService service = Mock()
        controller.catalogService = service
    and: "make sure it can file books"
        service.isAvailable(_ as Book) >> true

    when:"we verify isdn"
        params.isdn = book.isdn
        def result = controller.verifyISDN()

    then:"verify the book was filed"
        1 * service.inquired(_)
        result == true
}
```

Global

```
void "test interaction scoping"() {
    given:"create mock CatalogService"
        CatalogService service = Mock()
        controller.catalogService = service
    and: "make sure it can file books"
        service.isAvailable(_ as Book) >> true

    when:"we verify isdn"
        params.isdn = book.isdn
        def result = controller.verifyISDN()

    then:"verify the book was filed"
        1 * service.inquired(_)
        result == true
}
```

Local

```
void "test interaction scoping"() {
    given:"create mock CatalogService"
        CatalogService service = Mock()
        controller.catalogService = service
    and: "make sure it can file books"
        service.isAvailable(_ as Book) >> true

    when:"we verify isdn"
        params.isdn = book.isdn
        def result = controller.verifyISDN()

    then:"verify the book was filed"
        1 * service.inquired(_)
        result == true
}
```

Required

```
void "test interaction scoping"() {
    given:"create mock CatalogService"
        CatalogService service = Mock()
        controller.catalogService = service
    and: "make sure it can file books"
        service.isAvailable(_ as Book) >> true

    when:"we verify isdn"
        params.isdn = book.isdn
        def result = controller.verifyISDN()

    then:"verify the book was filed"
        1 * service.inquired(_)
        result == true
}
```

Optional

```
1 * catalogService.file(book, library) >> 'FILED'
```

Carnality

```
(0..3) * catalogService.file(book, library) >> 'FILED'
```

Carnality

```
(3..._) * catalogService.file(book, library) >> 'FILED'
```

Carnality

```
(_..3) * catalogService.file(book, library) >> 'FILED'
```

Carnality

`TooFewInvocationsError`

TooManyInvocationsError

```
(_..3) * catalogService.file(book, library) >> 'FILED'
```

Target Constraint

```
(_..3) * _.file(book, library) >> 'FILED'
```

Target Constraint

```
(_..3) * catalogService.file(book, library) >> 'FILED'
```

Method Constraints

```
(_..3) * catalogService./f.*/(book, library) >> 'FILED'
```

Method Constraints

```
(_..3) * catalogService._(book, library) >> 'FILED'
```

Method Constraints

```
(_..3) * catalogService.file(book, library) >> 'FILED'
```

Argument List Constraint

```
(_..3) * catalogService.file(_, _) >> 'FILED'
```

Argument List Constraint

```
(_..3) * catalogService.file(*_) >> 'FILED'
```

Argument List Constraint

```
(_..3) * catalogService.file(_ as Book, _) >> 'FILED'
```

Argument List Constraint

```
(_..3) * catalogService.file(_ as Book, !null) >> 'FILED'
```

Argument List Constraint

```
(_..3) * catalogService.file({it.isdn > 1}, !null) >> 'FILED'
```

Argument List Constraint

```
1 * catalogService.file(book, library) >> 'FILED'
```

Return Values

```
3 * catalogService.file(book, library) >>> ['FILED',‘ERROR’]
```

Return Values

```
3 * catalogService.file(book, library) >> {
                                            book.status = FILED
                                            return book
                                           }
```

Return Values

```
3 * catalogService.file(book, library) >> {
                        throw new TimeoutException()
                    }
```

Return Values

```
foo.bar() >> { throw new IOException() } >>> [1, 2, 3] >> { throw new
RuntimeException() }
```

Return Values

(_••_) * _•_(*_)

Order Is NOT enforced

```
void "test ordered interactions"() {
    . . .

    then:"verify the book was filed"
        1 * service.inquired(_)
        2 * service.notifyLibrarian(*_)
        result == true
}
```

Order is enforced

```
void "test ordered interactions"() {
    . . .

    then:"verify an inquiry was add to the book"
        1 * service.inquired(_)
   then:"verify that all librarians are notified "
        2 * service.notifyLibrarian(*_)
        result == true
}
```

Mocking Summary

- 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")
```

Interactions Summary

- Stubbing

```
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"]
```

Extensions

References

- Getting Spock
  - http://code.google.com/p/spock/
- Source from Presentation
  - https://github.com/kensipe/spock-javaone2014

Summary

- Closing and Q&A

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

JUnit Rules

SpockConfig