Are you Mocking Me?

with Spock

Ken Sipe
About Speaker

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 Philosophy
- Doubles and Fakes
- Spock Mocking
Testing
JUnit == Java Unit
Testing Philosophy
Class / Object
Package / Layers
Components
Infrastructure
Protocol
Topology
<table>
<thead>
<tr>
<th>Class / Object</th>
<th>AccountController</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package / Layers</td>
<td>AccountService</td>
</tr>
<tr>
<td>Components</td>
<td>AccountDAO</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>AccountDB</td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>Topology</td>
<td></td>
</tr>
</tbody>
</table>
If I have to test it all... if I have to “verify” the user experience..

then I will **test** all the way through.
The need to Fake it
Why Do We Fake it?
Reduce the setup overhead
Test Isolation
   Focus on a specific concern
Increase testing performance
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.

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

xUnit Test Patterns

Refactoring Test Code

Gerard Meszaros

Foreword by Martin Fowler
Test Double
Dummy

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

- Working implementations with built-in short-cuts
  - not production
  - in-memory database
- Stub
  - Objects with “canned” answers
  - Sometimes record interactions
Mock

- Objects with **pre-programmed** expectations
- Testing for behavior instead of state
<table>
<thead>
<tr>
<th>Type</th>
<th>Tool Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy</td>
<td>- developer only</td>
</tr>
<tr>
<td></td>
<td>- objenesis</td>
</tr>
<tr>
<td>Fake</td>
<td>- db options</td>
</tr>
<tr>
<td></td>
<td>- hibernate</td>
</tr>
<tr>
<td>Stubs</td>
<td></td>
</tr>
<tr>
<td>Mocks</td>
<td></td>
</tr>
</tbody>
</table>
- Classical TDD
- Mockist TDD
- Behavior Driven Development
  - off-shoot of mockist
Faking it with Spock
Spock Fakes

Spies
Stubs
Mocks
**Spock Mock**

- Response with:
  - zero
  - null
- is verified

```java
/**
 * 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

```java
/**
 * 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 Testing

Spock Mock

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

```java
/**
 * 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 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
}

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
}
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
}

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

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

cardinality
\((3..\_)*\) \.catalogService.file(\text{book, library}) \Rightarrow 'FILED'
(._..3) * catalogService.file(book, library) >> 'FILED'

cardinality
TooFewInvocationsError

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

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

Target Constraint
(3..3) * catalogService.file(book, library) >> 'FILED'

Target Constraint
(\_\_3) * catalogService./f.*/(book, library) >> 'FILED'

Method Constraints
(\_.3) * catalogService._(book, library) >> 'FILED'

Method Constraints
(__.3) * catalogService.file(book, library) >> 'FILED'

Method Constraints
(_.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.isbn > 1}, !null) >> 'FILED'

Argument List Constraint
1 * catalogService.file(book, library) >> 'FILED'

Argument List Constraint
3 * catalogService.file(book, library) >>> ['FILED', 'ERROR']

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

Return Values
catalogService.file(book, library) >> {
    throw new TimeoutException()
} >> 3 * catalogService.file(book, library) >>
foo.bar() >> { throw new IOException() } >>> [1, 2, 3] >> { throw new RuntimeException() }

Return Values
\[(\_\_\_\_\_) \times \_\_\_\_\_ (\_\_\_\_)\]
void "test ordered interactions"() {
    
    then:"verify the book was filed"
    1 * service.inquired(_)
    2 * service.notifyLibrarian(*_)
    result == true
}
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
}
Creating

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

Mocking

```java
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

- `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"]`
Getting Spock


Source from Presentation

- [https://github.com/kensipe/spock-demos-nfjs](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
Taxonomy of a Spec
import spock.lang.Specification

class MyFirstSpec extends Specification {
    // fields
    // fixture methods
    // feature methods
    // helper methods
}
 Specification

- compare to Test Case or Groovy Test Case
- Instructs JUnit to run with **Sputnik** (JUnit runner)

 Fields

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

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

- **Shared**
  - Setup once
  - `think setupSpec()`

- **statics**
  - `only use for constants`
Fixture Methods

```java
// 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

```python
// 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
<table>
<thead>
<tr>
<th>given:</th>
<th>preconditions, data fixtures</th>
</tr>
</thead>
<tbody>
<tr>
<td>when:</td>
<td>actions that trigger some outcome</td>
</tr>
<tr>
<td>then:</td>
<td>makes assertions about outcome</td>
</tr>
<tr>
<td>expect:</td>
<td>short alt to when &amp; then</td>
</tr>
<tr>
<td>where:</td>
<td>applies varied inputs</td>
</tr>
<tr>
<td>and:</td>
<td>sub-divides other blocks</td>
</tr>
<tr>
<td>setup:</td>
<td>alias for given</td>
</tr>
<tr>
<td>cleanup:</td>
<td>post-conditions, housekeeping</td>
</tr>
</tbody>
</table>
Common Blocks

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

- setup

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

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

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

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
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 "crazy math examples"() {
    when:
    def x = Math.max(1, 2)

    then:
    x == 2

    expect:
    Math.max(1, 2) == 2
}

- Expect Restrictions
  - conditions
  - variable defs
def "setup and cleanup example"() {
    setup:
    def file = new File("/some/path")
    file.createNewFile()

    // ...

cleanup:
    file.delete()
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

```java
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
GREAT INDIAN DEVELOPER SUMMIT 2019

Conference: April 23-26, Bangalore

Register early and get the best discounts!

www.developersummit.com  @greatindiandev  bit.ly/gidslinkedin