Test-Driven Development Practices in Java

Mockito Feature Deep-Dive

Mike Nolan
mnolanjr@gmail.com

pluralsight
hardcore developer training

Module Overview

- **Advanced Mockito concepts**
- **PowerMock framework**

Argument Matching

Argument Matching

-
```
OrderEntity orderEntityFixture = …

Mockito.when(mockOrderDao.findById(“23”)).thenReturn(orderEntityFixture);
```
  - Value types use the ‘==‘ operator

- **Explicit argument matchers provide flexibility**
  - These typically make the stub more generic
  - If any arguments are explicit, all must be explicit

```
List<OrderEntity> orderEntityListFixture = …

Mockito.when(mockOrderDao.findByCustomerId(Matchers.anyString()))
.thenReturn(orderEntityListFixture);
```

```
    List<OrderEntity> orders = orderDao.findByCustomerId(altCustomerRecord.getId());
    …
  }
  …
}
```

Argument Matching

- **Implicit argument matching**
  - Reference types use the equals(..) method
  - Value types use the '==' operator

- **Explicit argument matchers provide flexibility**
  - These typically make the stub more generic
  - If any arguments are explicit, all must be explicit

```
List<OrderEntity> orderEntityListFixture = …

Mockito.when(mockOrderDao.findByStateAndRegion("IL", Matchers.anyString()))
.thenReturn(orderEntityListFixture);
```

Argument Matching

- **Implicit argument matching**
  - Reference types use the equals(..) method
  - Value types use the '==' operator

- **Explicit argument matchers provide flexibility**
  - These typically make the stub more generic
  - If any arguments are explicit, all must be explicit

```
List<OrderEntity> orderEntityListFixture = …

Mockito.when(mockOrderDao.findByStateAndRegion(Matchers.eq("IL"), Matchers.anyString()))
.thenReturn(orderEntityListFixture);
```

Matchers

- **Matchers.eq(..)**
- **Any matchers**

```
Mockito.when(mockOrderDao.findById(Matchers.anyInt())).thenReturn(..);
Mockito.when(mockCalculator.multiply(Matchers.anyDouble())).thenReturn(..);

Mockito.when(mockOrderDao.findByName((String) Matchers.any())).thenReturn(..);
Mockito.when(mockOrderDao.findByName(Matchers.any(String.class))).thenReturn(..);

Mockito.when(mockOrderDao.findByParams((Map<String, String>) Matchers.anyMap()))
.thenReturn(..);
Mockito.when(mockOrderDao.findByParams(Matchers.anyMapOf(String.class, String.class)))
.thenReturn(..);
```

Matchers

- **Matchers.eq(..)**
- **Any matchers**
- **String matchers**

```
Mockito.when(mockOrderDao.findByName(Matchers.eq(“Brown”))).thenReturn(..);

Mockito.when(mockOrderDao.findByName(Matchers.contains(“own”))).thenReturn(..);

Mockito.when(mockOrderDao.findByName(Matchers.startsWith(“Br”))).thenReturn(..);
Mockito.when(mockOrderDao.findByName(Matchers.endsWith(“wn”))).thenReturn(..);

Mockito.when(mockOrderDao.findByName(Matchers.match(“^(Br|Cr)own”))).thenReturn(..);
```

Matchers

- **Matchers.eq(..)**
- **Any matchers**
- **String matchers**
- **Reference equality and reflection**
  - Test reference equality with *Matchers.same(ref)*
  - Reflectively test using
    - *Matchers.refEq(ref)*
    - *Matchers.refEq(ref, “excludeField”)*

Stubbing Consecutive Calls

Stubbing Consecutive Calls

- **Handy for testing logic that needs to be resilient when errors occur**
  1. Looping logic that either short-circuits or continues on exception
  2. Retry logic when errors are encountered

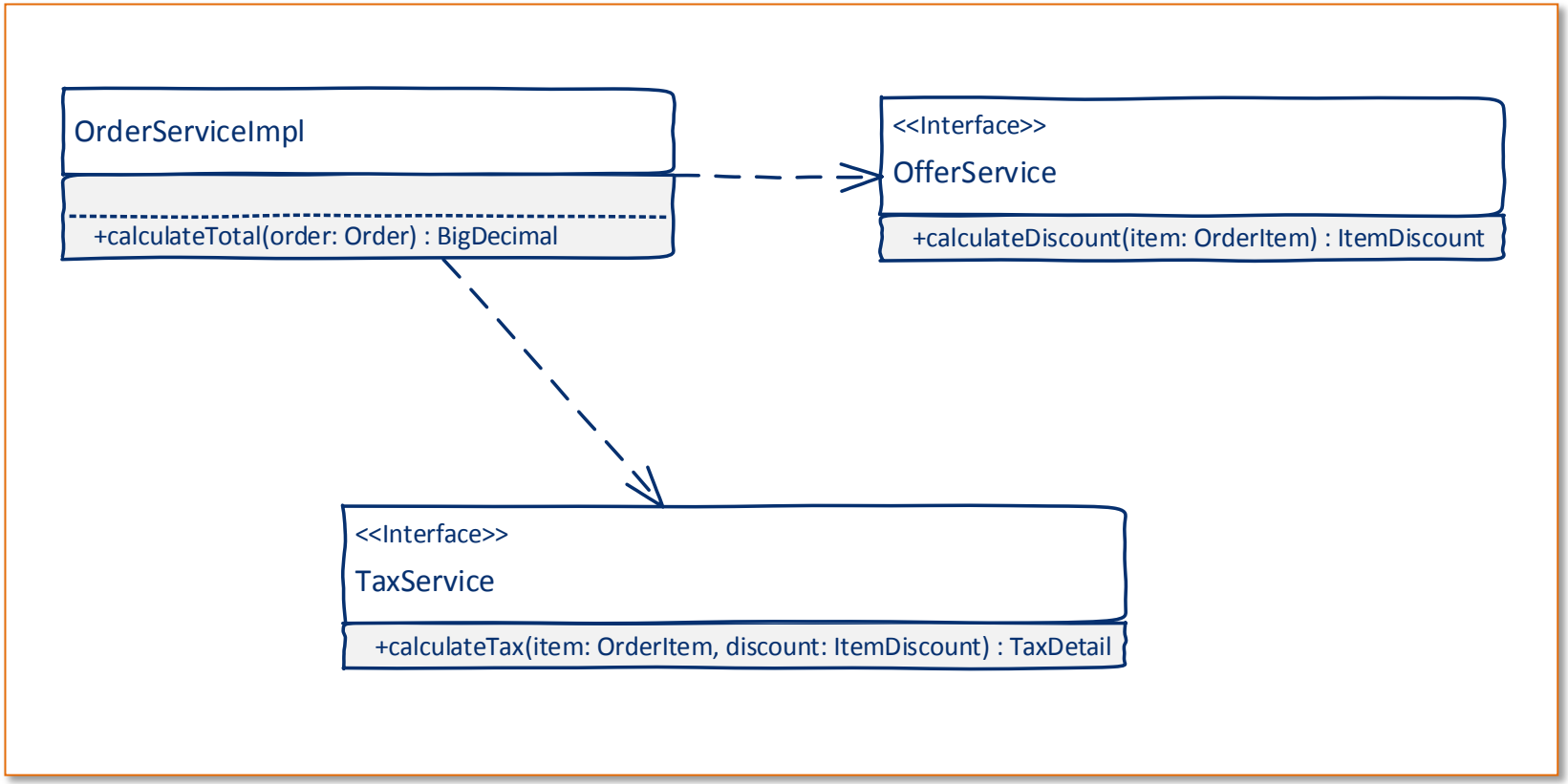
- **Stubbing consecutive responses helps simplify these types of tests**

Verification Order

Sometimes Ordering Is Critical

- **Legacy APIs and dependencies sometimes enforce restrictions**
- **Results of one dependency may be needed when interacting with another**

InOrder Verifier



InOrder Verifier

```
// Setup
OrderItem orderItemFixture = new OrderItem();  // Then set remaining data
ItemDiscount itemDiscountFixture = new ItemDiscount();
itemDiscountFixture.setTaxableDiscount(false);

Mockito.when(mockOfferService.calculateDiscount(orderItemFixture))
.thenReturn(itemDiscountFixture);

TaxDetail taxAmountFixture = new TaxDetail("2.34");

Mockito.when(mockTaxService.calculateTax(orderItemFixture, itemDiscountFiture)
.thenReturn(taxAmountFixture);

// Execute method under test

// Verification
InOrder inOrderVerifier = Mockito.inOrder(mockOfferService, mockTaxService);

inOrderVerifier.verify(mockOfferService).calculateDiscount(orderItemFixture);
inOrderVerifier.verify(mockTaxService).calculateTax(orderItemFixture, itemDiscountFixture);
```

InOrder Verifier

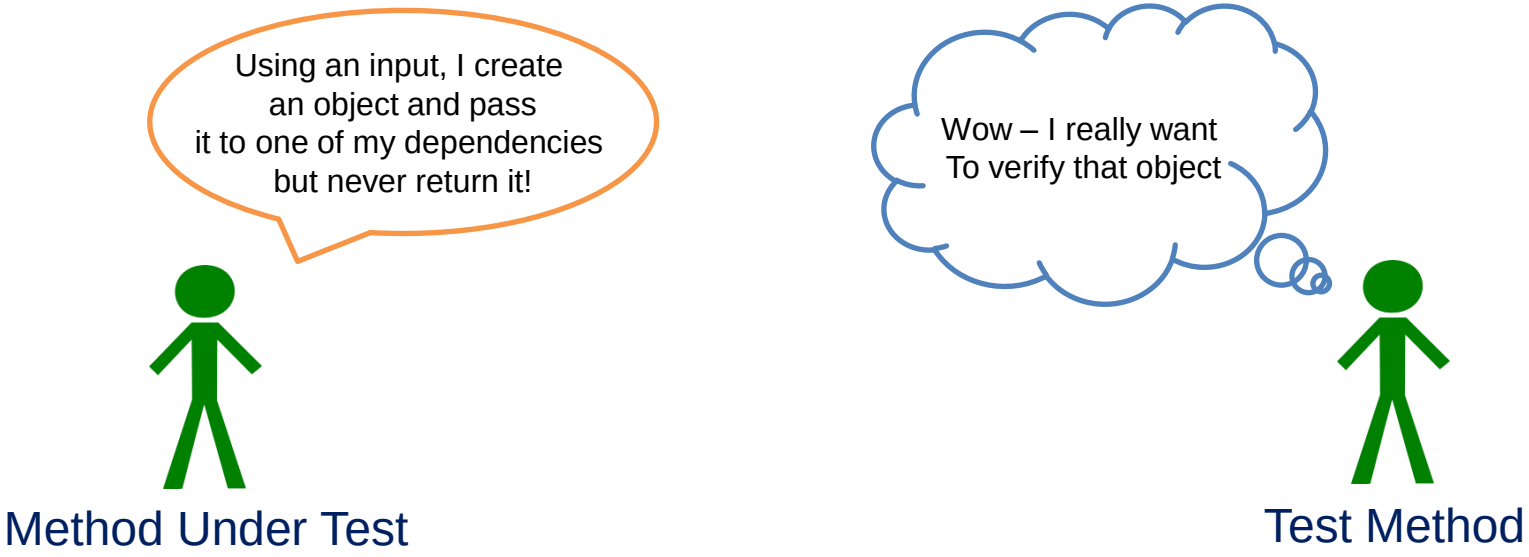
```
// Setup
OrderItem orderItemFixture = new OrderItem();  // Then set remaining data
ItemDiscount itemDiscountFixture = new ItemDiscount();
itemDiscountFixture.setTaxableDiscount(false);

Mockito.when(mockOfferService.calculateDiscount(orderItemFixture))
.thenReturn(itemDiscountFixture);

TaxDetail taxAmountFixture = new TaxDetail("2.34");

Mockito.when(mockTaxService.calculateTax(orderItemFixture, itemDiscountFiture)
.thenReturn(taxAmountFixture);

// Execute method under test

// Verification
InOrder inOrderVerifier = Mockito.inOrder(mockOfferService, mockTaxService);

inOrderVerifier.verify(mockOfferService).calculateDiscount(orderItemFixture);
inOrderVerifier.verify(mockTaxService).calculateTax(orderItemFixture, itemDiscountFixture);
```

InOrder Verifier

```
// Setup
OrderItem orderItemFixture = new OrderItem();  // Then set remaining data
ItemDiscount itemDiscountFixture = new ItemDiscount();
itemDiscountFixture.setTaxableDiscount(false);

Mockito.when(mockOfferService.calculateDiscount(orderItemFixture))
.thenReturn(itemDiscountFixture);

TaxDetail taxAmountFixture = new TaxDetail("2.34");

Mockito.when(mockTaxService.calculateTax(orderItemFixture, itemDiscountFiture)
.thenReturn(taxAmountFixture);

// Execute method under test

// Verification
InOrder inOrderVerifier = Mockito.inOrder(mockOfferService, mockTaxService);

inOrderVerifier.verify(mockOfferService).calculateDiscount(orderItemFixture);
inOrderVerifier.verify(mockTaxService).calculateTax(orderItemFixture, itemDiscountFixture);
```

Capturing Arguments

Validating Objects Without Direct Access



Argument Captors

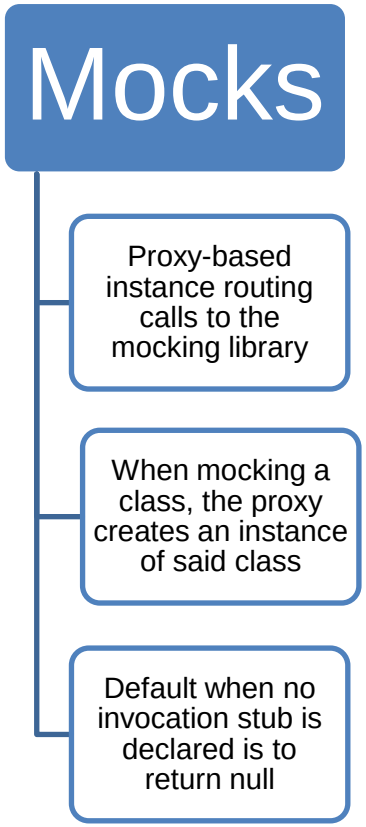
Capturing Arguments

- **Capturing allows you to capture the actual object passed into the mock**
- **ArgumentCaptor instances are used to capture these values**

Partial Mocks

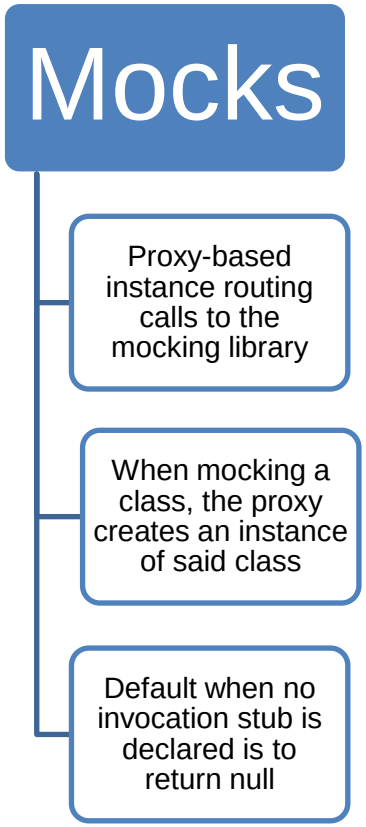
Mocks vs. Spies

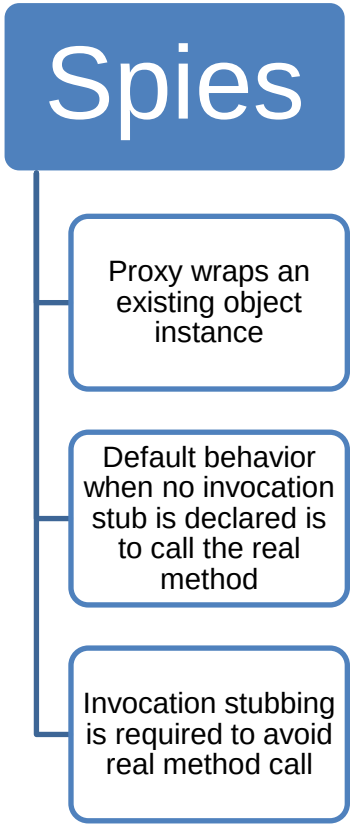
- **Mocking Interfaces vs. Classes**
- **Partial mocking mixes controlled invocation stubs with real method calls**



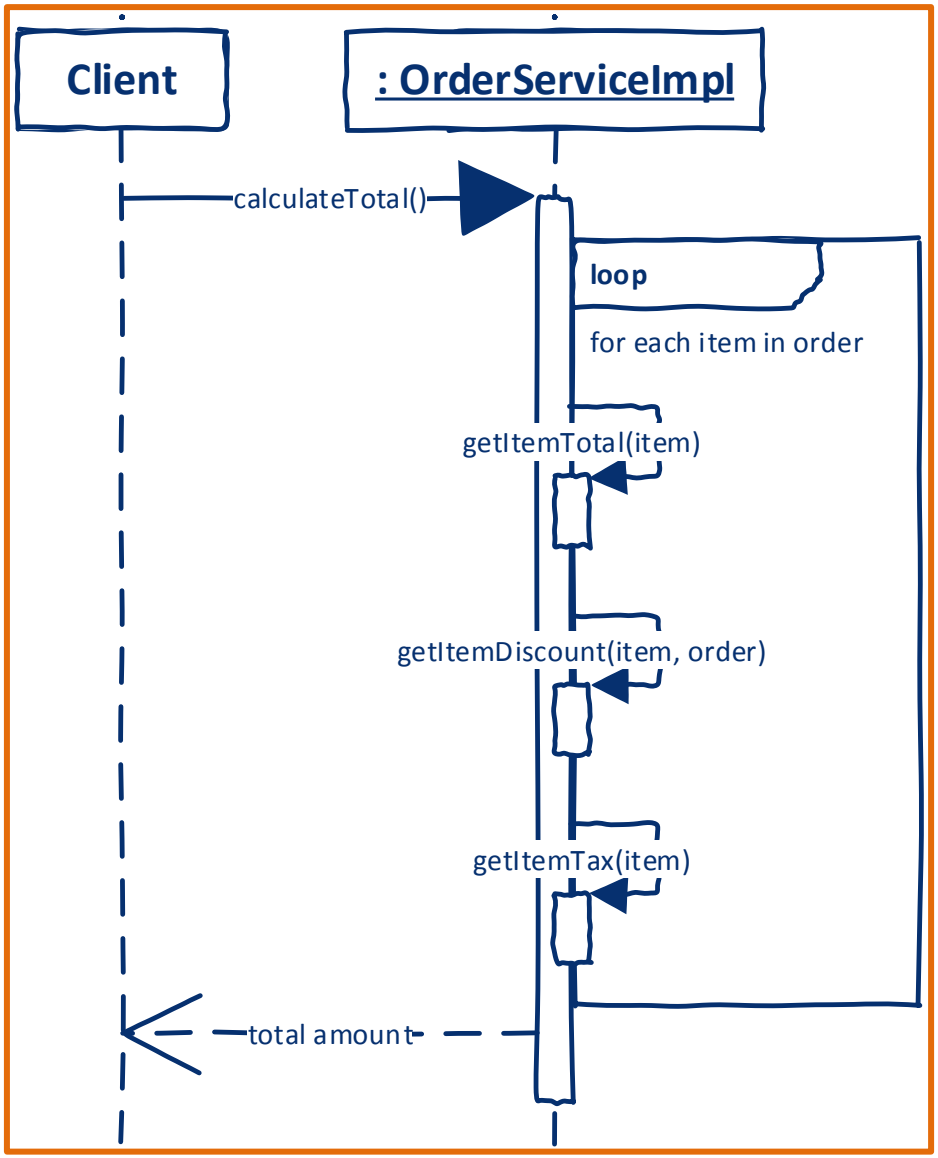
Mocks vs. Spies

- **Mocking Interfaces vs. Classes**
- **Partial mocking mixes controlled invocation stubs with real method calls**





Partial Mocks – Simplify Testing



Things To Be Mindful Of With Partial Mocks

- **When partial mocking, please bear the following in mind**
  - You can’t mock final methods
  - You can’t mock private methods
  - Set the state appropriately

- **When stubbing a spy, the initial call is routed to the real method**

Things To Be Mindful Of With Partial Mocks

- **When partial mocking, please bear the following in mind**
  - You can't mock final methods
  - You can't mock private methods
  - Set the state appropriately

- **When stubbing a spy, the initial call is routed to the real method – this can result in unexpected exceptions**

Things To Be Mindful Of With Partial Mocks

- **When partial mocking, please bear the following in mind**
  - You can't mock final methods
  - You can't mock private methods
  - Set the state appropriately

- **When stubbing a spy, the initial call is routed to the real method – this can result in unexpected exceptions**

```
List<String> liveList = new LinkedList<String>();
List<String> spyList = Mockito.spy(liveList);

// This will result in an IndexOutOfBoundsException !!!
Mockito.when(spyList.get(0)).thenReturn("A string result");
```

Things To Be Mindful Of With Partial Mocks

- **When partial mocking, please bear the following in mind**
  - You can't mock final methods
  - You can't mock private methods
  - Set the state appropriately

- **When stubbing a spy, the initial call is routed to the real method – this can result in unexpected exceptions**

```
List<String> liveList = new LinkedList<String>();
List<String> spyList = Mockito.spy(liveList);

// This will now work successfully!
spyList.add("A dummy value");
Mockito.when(spyList.get(0)).thenReturn("A string result");
```

After the stub, a call of spyList.get(0) would return "A string result"

When Mockito Is Not Enough…

PowerMock

- **Mockito covers 80% of your usage scenarios**
- **PowerMock provides an extension for the remaining 20%**
- **The difference is**
  - Mockito uses a proxy-based approach to intercept calls
  - PowerMock uses a custom class loader and manipulates the byte code

Mocking Static Method Invocations

Using PowerMock & Static Method Stubbing

- **@RunWith(PowerMockRunner.class)**
- **@PrepareForTest(value={ClassToInstrument.class})**
- **PowerMockito.mockStatic(ClassWithStaticMethods.class)**
- **PowerMockito.when(*Class.staticMethod(arg)*).then(*value*)**
- **PowerMockito.verifyStatic()**

Replacing Object Instantiation

Replacing Object Instantiation

- **Calls using ‘new’ operator can be replaced with stubbed results**
- **PowerMockito.whenNew(..)**
  - Call zero parameter constructor by class name
  - Use reflection for specific constructor
  - Specify specific constructor using a string value
- **whenNew(..) returns PowerMock’s version of OngoingStub<T>**
  - ConstructorExpectationSetup<T>
  - WithOrWithoutExpectedArguments<T>
- **@PrepareForTest(*ClassUnderTest.class*)**
  - Specify the class under test, not the class being instantiated

Stubbing Final & Private Methods

Final & Private Methods

- **PowerMockito.mock(..)**
- **Simply using a PowerMockito mock allows final methods to be stubbed**
- **A specific overload of PowerMockito.when(..) allows private method mocking**

Stubbing Private Methods

- **Pass the mock and a Java Reflection Method object into the when method & WithOrWithoutExpectedArguments is returned**

```
OrderServiceImpl mockOrderService = Mockito.mock(OrderServiceImpl.class);

Mockito.when(mockOrderService.calculateTotal(order)).thenCallRealMethod();

BigDecimal discountResult = new BigDecimal("1.50");
Method calculateDiscount = …

WithOrWithoutExpectedArguments args = Mockito.when(mockOrderService, calculateDiscount);
args.withArguments(item1, order).thenReturn(discountResult);
```

Verifying Private Methods

- **PowerMockito.verifyPrivate(..) supports several overloads**
  - PrivateMethodVerification is returned
- **PrivateMethodVerification.invoke(..) supports several overloads to verify the call during the test**
  - Leverage the Java Reflection Method object to verify & returns WithOrWithoutVerifiedArguments
  - Pass a string value containing the method name, allowing with the arguments via a vararg parameter
  - Final version simply takes arguments – I don't recommend this

```
BigDecimal discountResult = new BigDecimal("1.50");
Method calculateDiscount = …

PrivateMethodVerification pmVerification = Mockito.verifyPrivate(mockOrderService);
// Use either this
pmVerification.invoke(method).withArguments(item1, order);

// Or this
pmVerification.invoke("calculateDiscount", item1, order);
```

Whitebox Test Utility Class

Whitebox

- **Wrapper around Java Reflection API, but geared towards testing**
  - Good for testing private methods
  - Removes exception handling – the test will fail if encountered during Whitebox method calls

- **Art of the possible with Whitebox**
  - Find an objects constructors, methods, and fields
  - Invoke methods & constructors
  - Get & set field values

Summary

- **Mockito advanced features**
  - Argument matchers
  - Stubbing consecutive calls
  - Verification order
  - Argument capturing
  - Spying & Partial Mocks

- **PowerMock**
  - Stubbing static method calls
  - Replacing object instantiation with constructor stubbing
  - Mocking final & private methods
  - Whitebox utility class