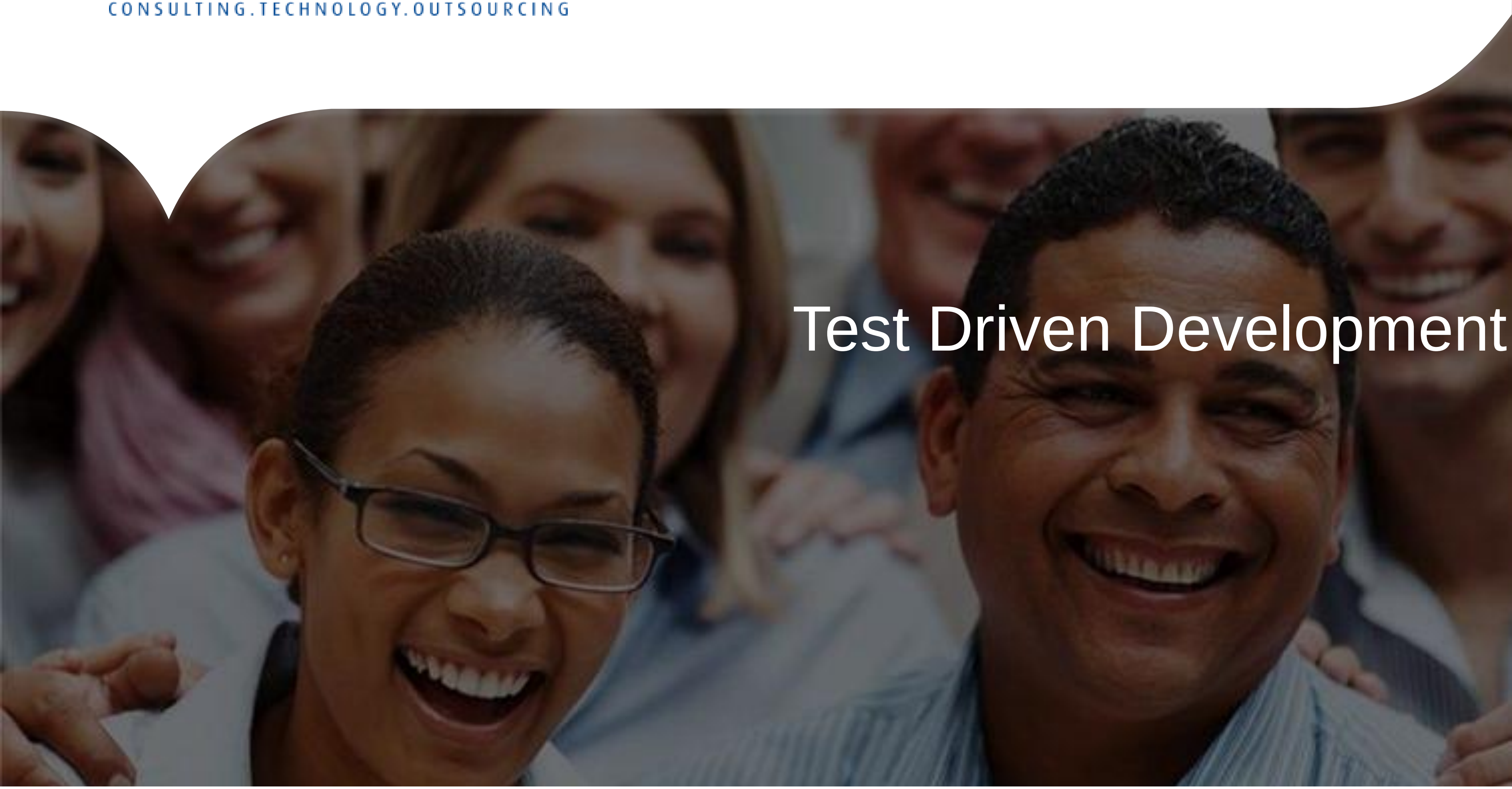
CONSULTING.TECHNOLOGY.OUTSOURCING



Test Driven Development

People matter, results count.

Recap

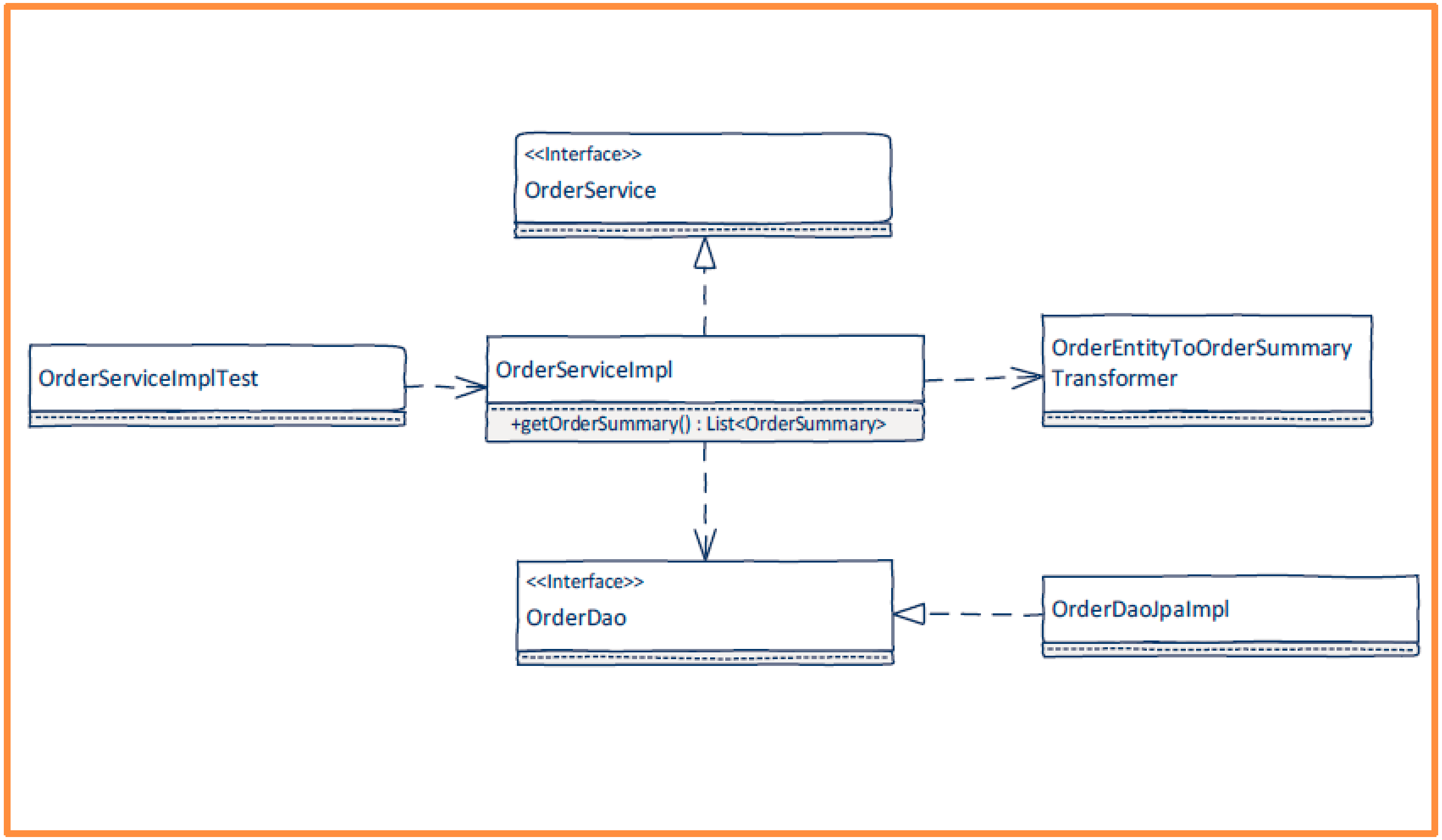
- ➢Unit Testing
- ➢Objectives of Unit Testing
- ➢Planning
- ➢Unit Test Design
- ➢Unit Testing implementation using Junit
- ➢Best Practices of Unit Testing

Agenda

- Mockito
- Advanced Mockito Concepts
- Power Mockito
- Test Driven Development

Mockito

Mocking Concepts

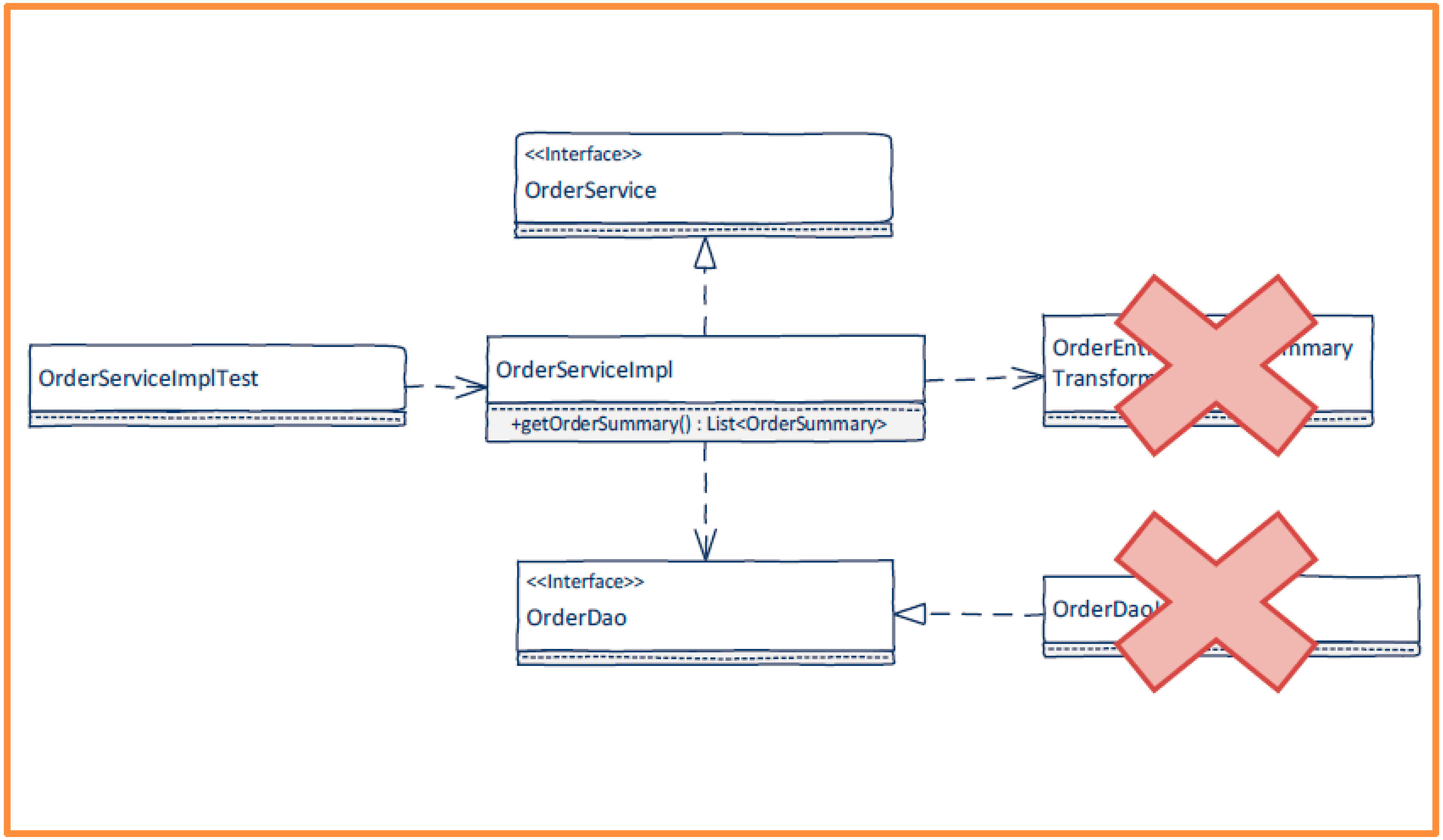


Mocking Concepts

- **Methods under test often leverage dependencies**
- **Testing with dependencies creates challenges**
    - Live database needed
    - Multiple developers testing simultaneously
    - Incomplete dependency implementation

- **Mocking frameworks give you control**

Mocking Concepts



Mocking Options

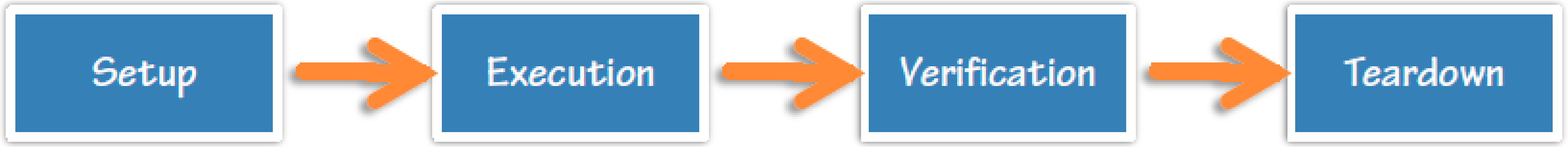
- **Implement the mocked functionality in a class**
    - This approach is tedious and obscure

- **Leverage a mocking framework**
    - Avoid class creation
    - Leverages the proxy pattern

- **Multiple options –Mockito, EasyMock, JMock**

Mockito Overview

Support unit testing cycle



Setup –Creating the mock
OrderDao mockOrderDao= Mockito.*mock(OrderDao.class)*

Setup –Method stubbing
Mockito.*when(mockOrderDao.findById(idValue)).thenReturn(orderFixture)*

Verification
Mockito.*verify(mockOrderDao).findById(idValue)*

Creating Mock Instances

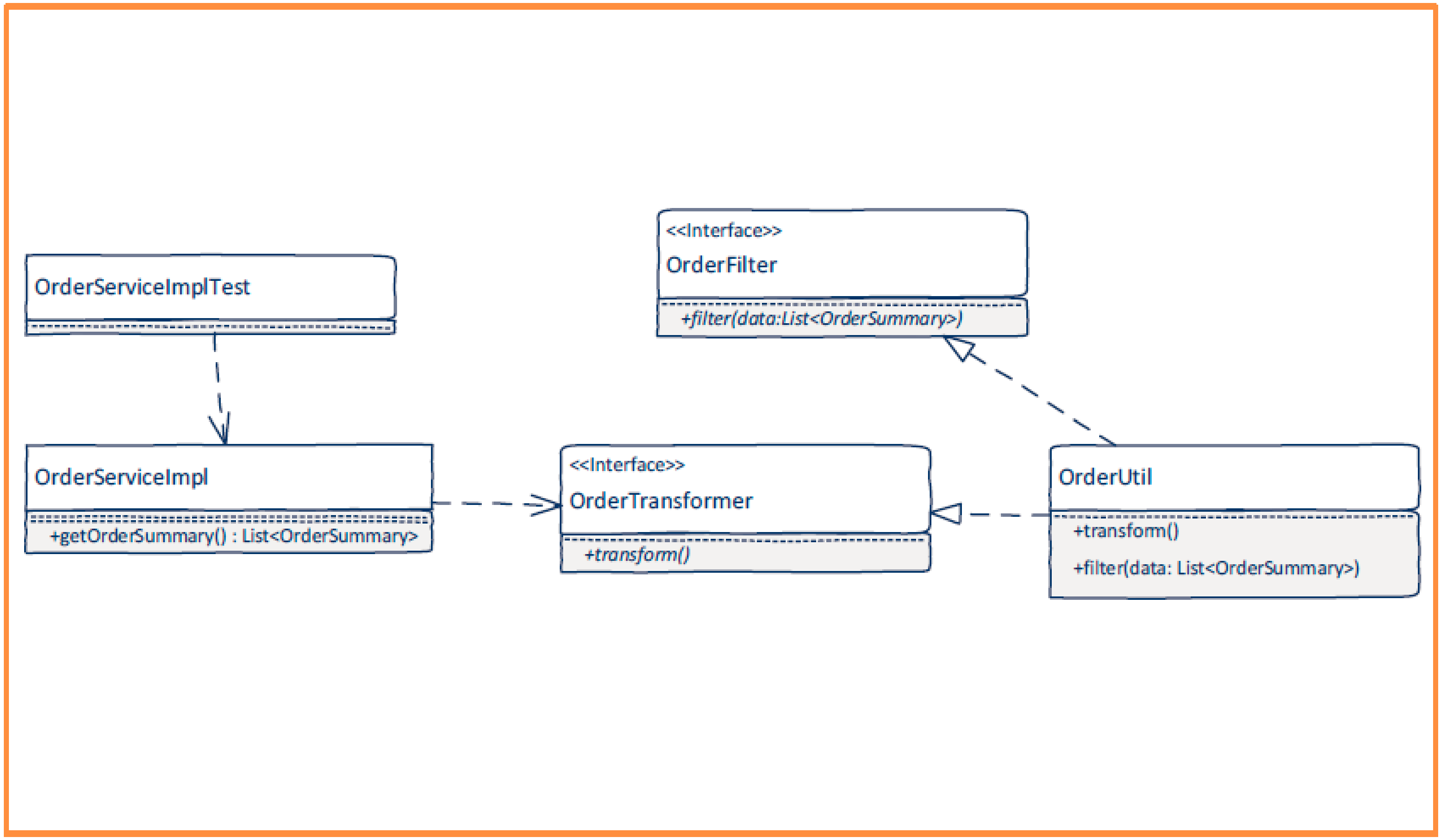
- Mockito.mock(*Class<?> class) is the core method for creating mocks*
    - @Mock is an alternative

Mock Settings

•The MockSettingsinterface provides added control for mock creation

•Use MockSettings.extraInterfaces(..) to add interfaces supported by

Mock Settings



Mock Settings

- **The MockSettingsinterface provides added control for mock creation**
- **Use MockSettings.extraInterfaces(..) to add interfaces supported by the mock**
- **MockSettings.serializable() creates a mock which can be bassedas a serializableobject**
- **MockSettings.name(..) specifies a name when verification of the mock fails**

Stubbing Method Calls

- **Provides capability to define how method calls behave via when/then pattern**
- **Calling Mockito.when(..) returns OngoingStub<T>, specifying how the invocation behavesthenReturn(..)**
    - thenThrow(..)
    - thenCallRealMethod(..)
    - thenAnswer(..)

Stubbing Method Calls

void Methods

- Mocking void methods do not work with OngoingStub<T>
- Mockito.doThrow(..) returns the Stubberclass

Verifications

Mockito.verify(..) is used to verify an intended mock operation was called

- **Mockito.verify(..) is used to verify an intended mock operation was called**
- **VerificationModeallows extra verification of the operation**

Verifications

- **Mockito.verify(..) is used to verify an intended mock operation was called**
- **VerificationModeallows extra verification of the operationtimes(*n*)**
  - atLeastOnce()
  - atLeast(*n*)
  - atMost(*n*)
  - never()

- **Verifying no interactions globallyMockito.verify(..).zeroInteractions()**
  - Mockito.verify(..).noMoreInteractions()

Summary

Mocking Concepts

Mockito Basic Features

Set up

Verification

Advanced Mockito Concepts

- **Argument matching**
- **Matchers**

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

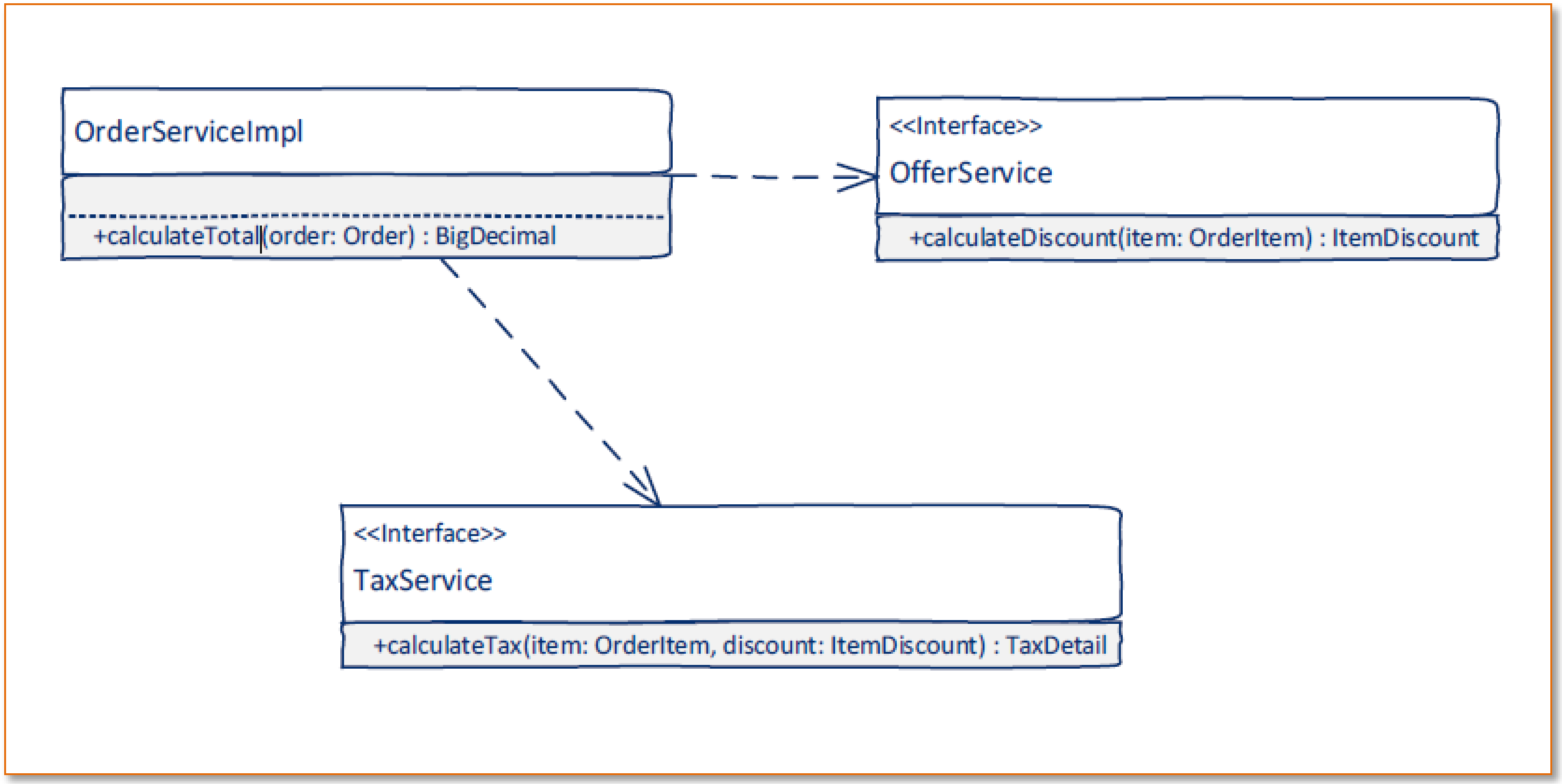
- **Handy for testing logic that needs to be resilient when errors occur**
  - 1.Looping logic that either short-circuits or continues on exception
  - 2.Retry logic when errors are encountered
- **Stubbing consecutive responses helps simplify these types of tests**

Verification Order

Sometimes Ordering Is Critical

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

In Order Verifier


OrderServiceImpl
+calculateTotal(order: Order) : BigDecimal
<<Interface>>
OfferService
+calculateDiscount(item: OrderItem) : ItemDiscount
<<Interface>>
TaxService
+calculateTax(item: OrderItem, discount: ItemDiscount) : TaxDetail

Capturing Arguments

Validating Objects Without Direct Access

Using an input, I create an object and pass it to one of my dependencies but never return it!
Wow – I really want To verify that object
Method Under Test
Test Method

Argument Captors

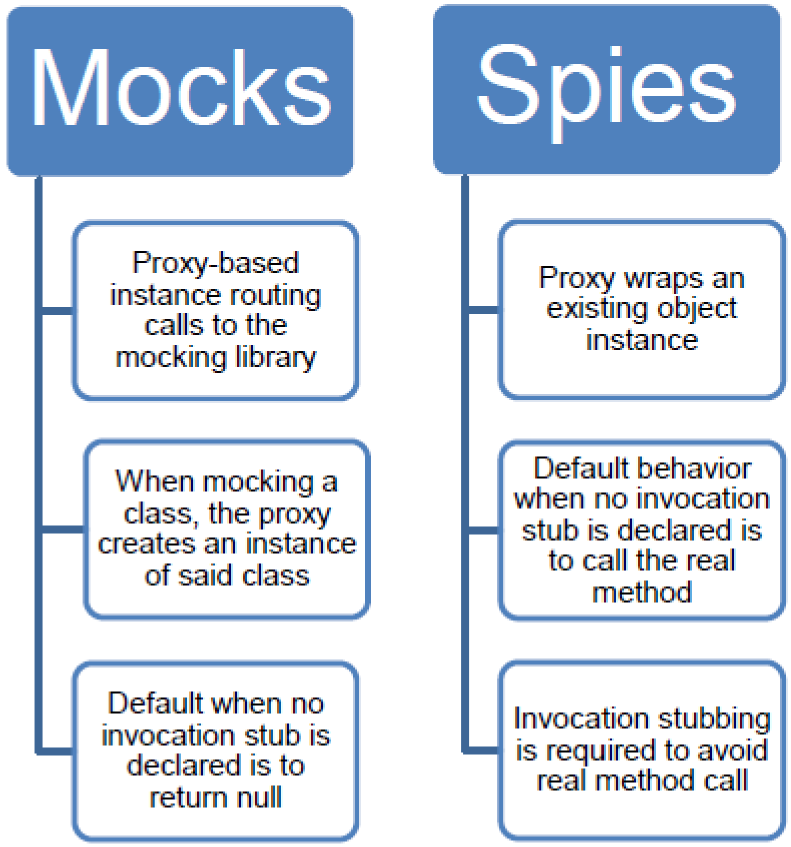
Capturing Arguments

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

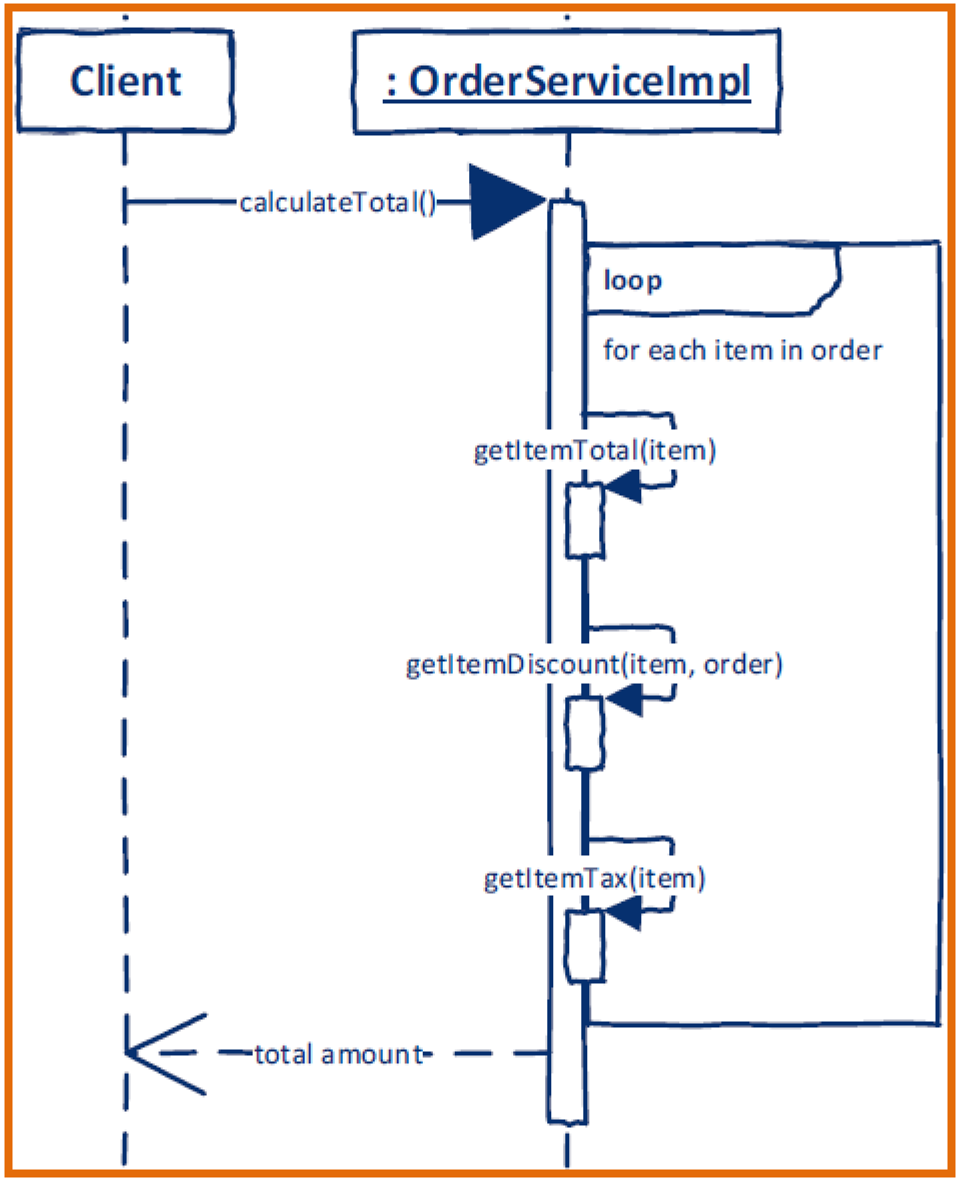
Partial Mocks

Mocks vs. Spies

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


Mocks
Proxy-based instance routing calls to the mocking library
When mocking a class, the proxy creates an instance of said class
Default when no invocation stub is declared is to return null
Spies
Proxy wraps an existing object instance
Default behavior when no invocation stub is declared is to call the real method
Invocation stubbing is required to avoid real method call

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**

Power Mockito

When Mockito Is Not Enough…

Power Mock

- **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

- **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

- **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

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

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

Test Driven Development Best Practices

Integrating Automation Suite within Source Control

- **Source Control:**
  - **GIT**
    - **Take latest test code from Source Control System and generate test reports.**

Approach towards Defect Management, Metrics and Reporting

- **Analyze the Test Reports**
- **Show Metrics with all Defects**
- **Generate Reports**
- **Send reports to relevant team members**

Summary

Mockito advanced features

Stubbing

PowerMock

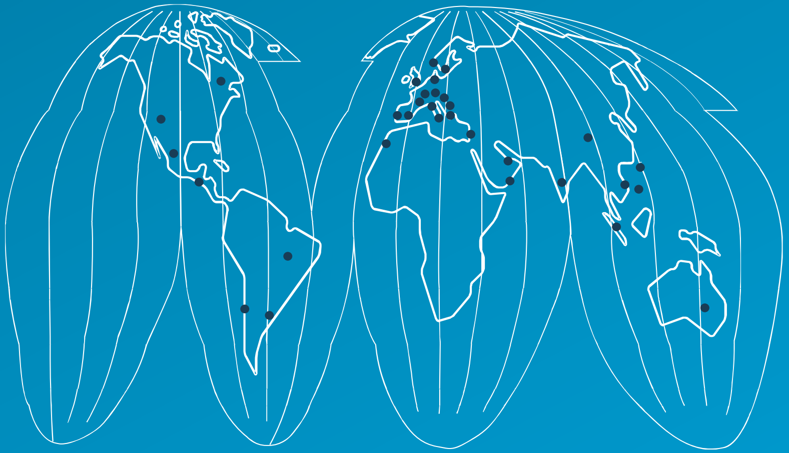
Mocking final & private methods

People matter, results count.

About Capgemini

With more than 145,000 people in 40 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2014 global revenues of EUR 10.5 billion.
Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini



www.capgemini.com