Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

Test Driven Development

People matter, results count.

Ground Rules for Face-to-face Classrooms

Everyone participates

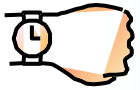
Respect individual opinions and diversities

Be open and honest

Give headlines, be concise

One speaker at a time

Make language a non-issue



Stick to time contracts

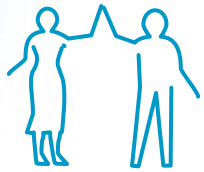
Seek first to understand, and then to be understood

Clean desk / room policy

No mobile phone,
No computer (except for surveys, polls etc)

Clients & Leadership are often around, please remember that

Maintain a spirit of fun and enthusiasm



Ground Rules for Virtual Classrooms

Participate actively in each session

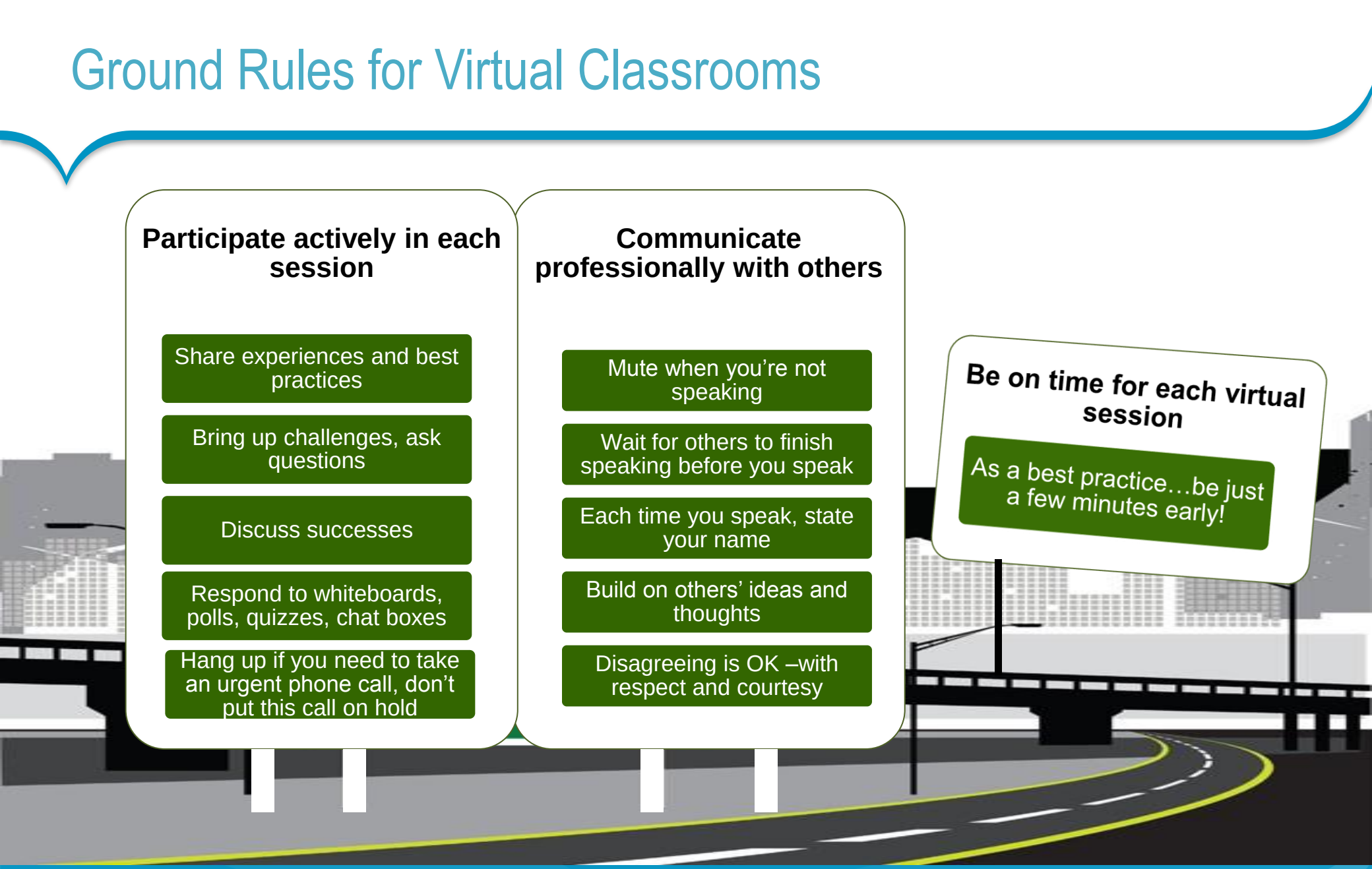
- Share experiences and best practices
- Bring up challenges, ask questions
- Discuss successes
- Respond to whiteboards, polls, quizzes, chat boxes
- Hang up if you need to take an urgent phone call, don't put this call on hold

Communicate professionally with others

- Mute when you're not speaking
- Wait for others to finish speaking before you speak
- Each time you speak, state your name
- Build on others' ideas and thoughts
- Disagreeing is OK –with respect and courtesy

Be on time for each virtual session

As a best practice…be just a few minutes early!



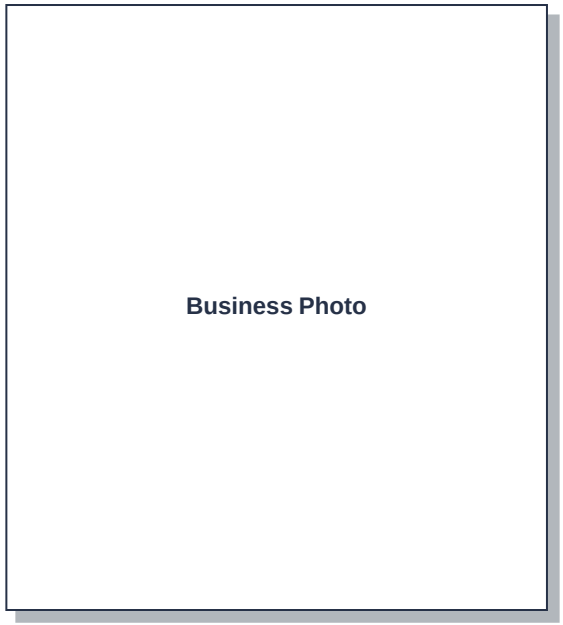
Module at a Glance

SME to provide the details required in the table.

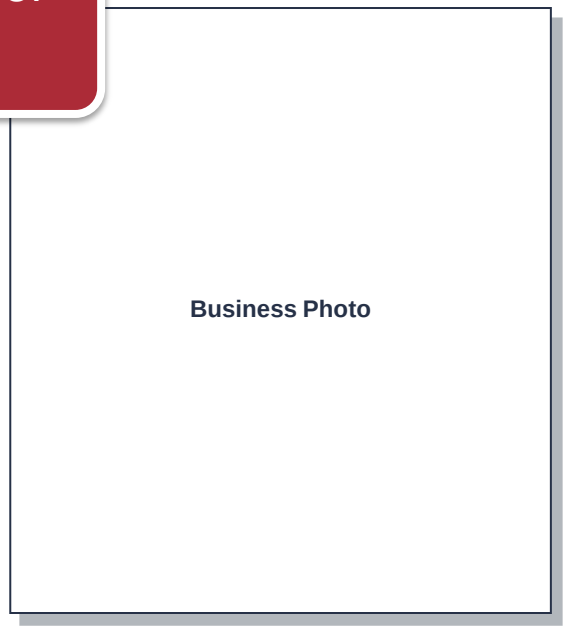
Target Audience:	
Course Level:	*Expert*
Duration (in hours):	8 hrs
Pre-requisites, if any:	Java
Post-requisites, if any:	*Submit Session Feedback*
Relevant Certifications:	None

Introductions (for Virtual Classrooms)

SME to provide the photos and names of the facilitators.

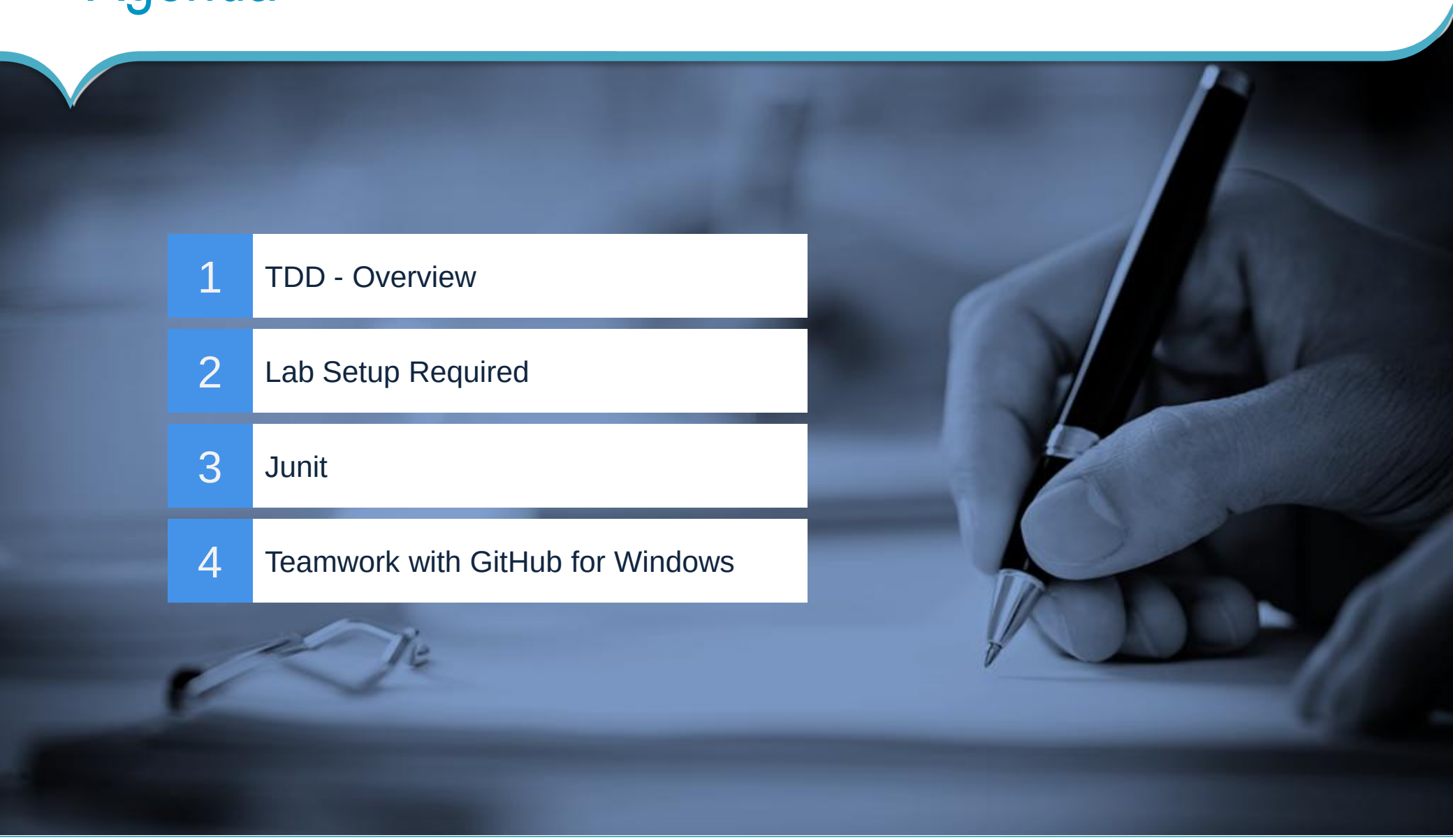


Facilitator
Vijayalakshmi David
Sr Consultant



Moderator
Name
Role

Agenda



1. TDD - Overview
2. Lab Setup Required
3. Junit
4. Teamwork with GitHub for Windows

Module Objectives

Note to the SME : Please provide the module Objectives or validate the partially updated content

What you will learn

At the end of this module, you will learn:

- What is TDD

What you will be able to do

At the end of this module, you be able to:

- TDD
- Junit
- Mockito

TDD - Overview

Outline

1. What is Test-Driven Development (TDD)
2. Problems with the traditional approach
3. How you benefit
4. Red / Green /Refactor
5. Development Task Focus
6. Terminology
7. Common Myths About TDD
8. Testing Framework

What is Test-Driven Development (TDD)?

- Development approach keeping tests one step ahead of your code
- Test-Driven Development →Test-Driven Design
- Early practices established anti-patterns

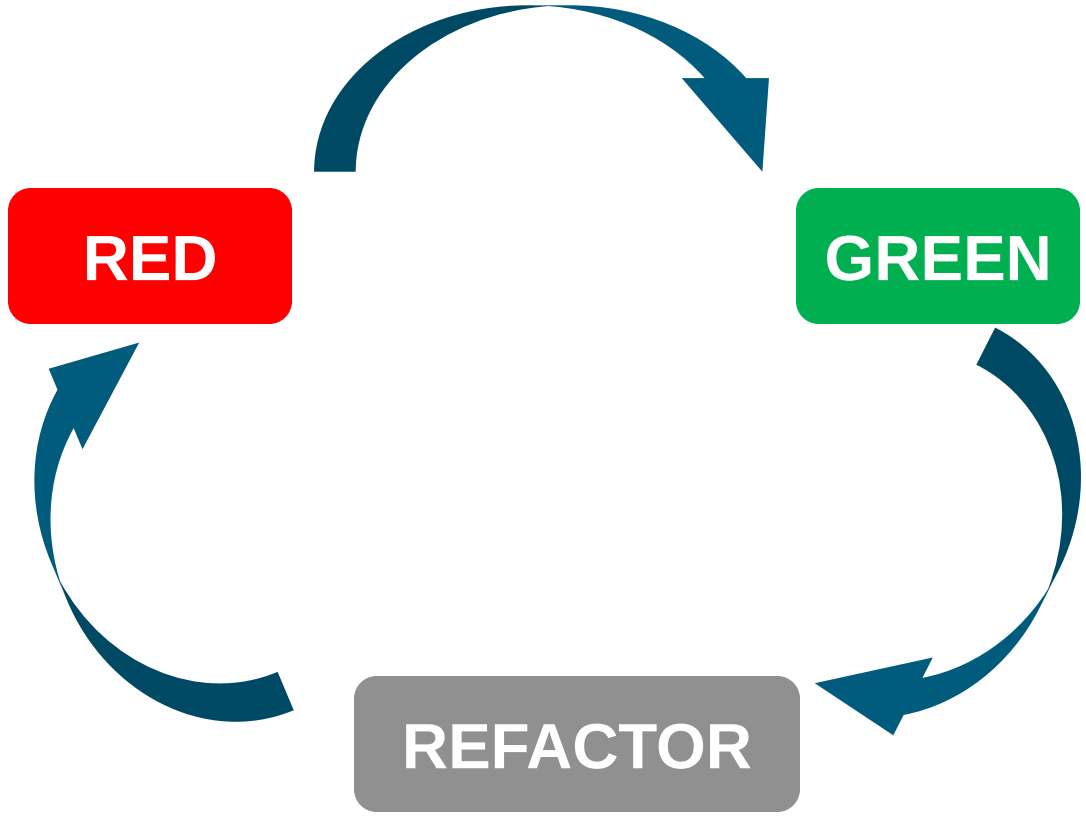
Problems with the traditional approach

- **Omission of thought in automated unit testing approach**
- **Benefits of automation missed**
- **Time crunch encountered in projects**
- **Mountain of work to implement the automated unit test**

How you benefit

- Balance between automated testing and functional coding during development sprints
- Consistent repeatability of test execution
- Up-front design with testing in mind
- More even gauge of progress between testing and coding

Red / Green / Refactor

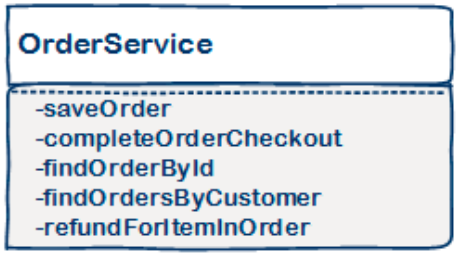


Development Task Focus

Create functional slice to add Item to an order
(focus on all layers for single story)

Vs.

Create all operations of Order Service
(focus on single layer)



Terminology

- **xUnit Testing**
- **Class-Under-Test**
- **Method-Under-Test**
- **Test Fixture**

Myth –Coding effort increases significantly

- **“Doesn’t all this testing double the development?”**
- **Question you should be asking –“If you are not automating your unit tests, then how are you executing the unit tests in a repeatable and consistent manner?”**
- **Over time, you will never manually unit test all cases in a repeatable manner with less effort than creating the test up-front; you are sacrificing quality**
- **The following link contains some analysis and calculations –**

  http://c2.com/cgi/wiki?UnitTestingCostsBenefits

Myth –All tests are written before any code

- **How do I write all my tests before I've written a line of functional code?**

- **You don't!**
  - **Follow red/green/refactor**
  - Stub out methods for important conditions
  - Add tests as bugs are discovered

Testing Framework

- **Mockito**
- **DBUnit**
- **PowerMockito**

Summary

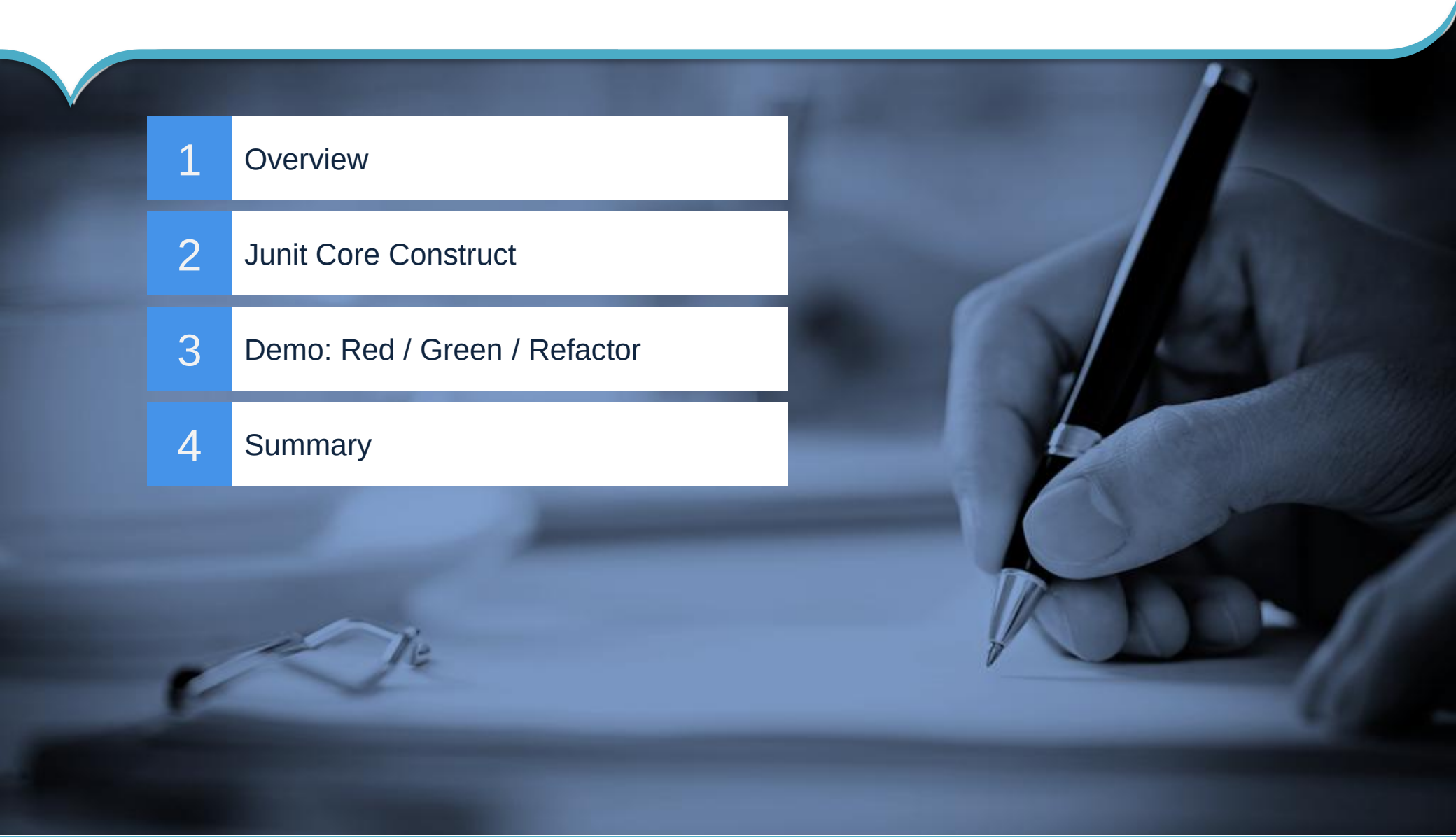
- **Basic principles & benefits of Test-Driven Development**
- **Basic tenants of the Red/Green/Refactor approach**
- **Explored common myths**
- **Brief framework introduction**

Lab Setup Required

Required Software for Lab

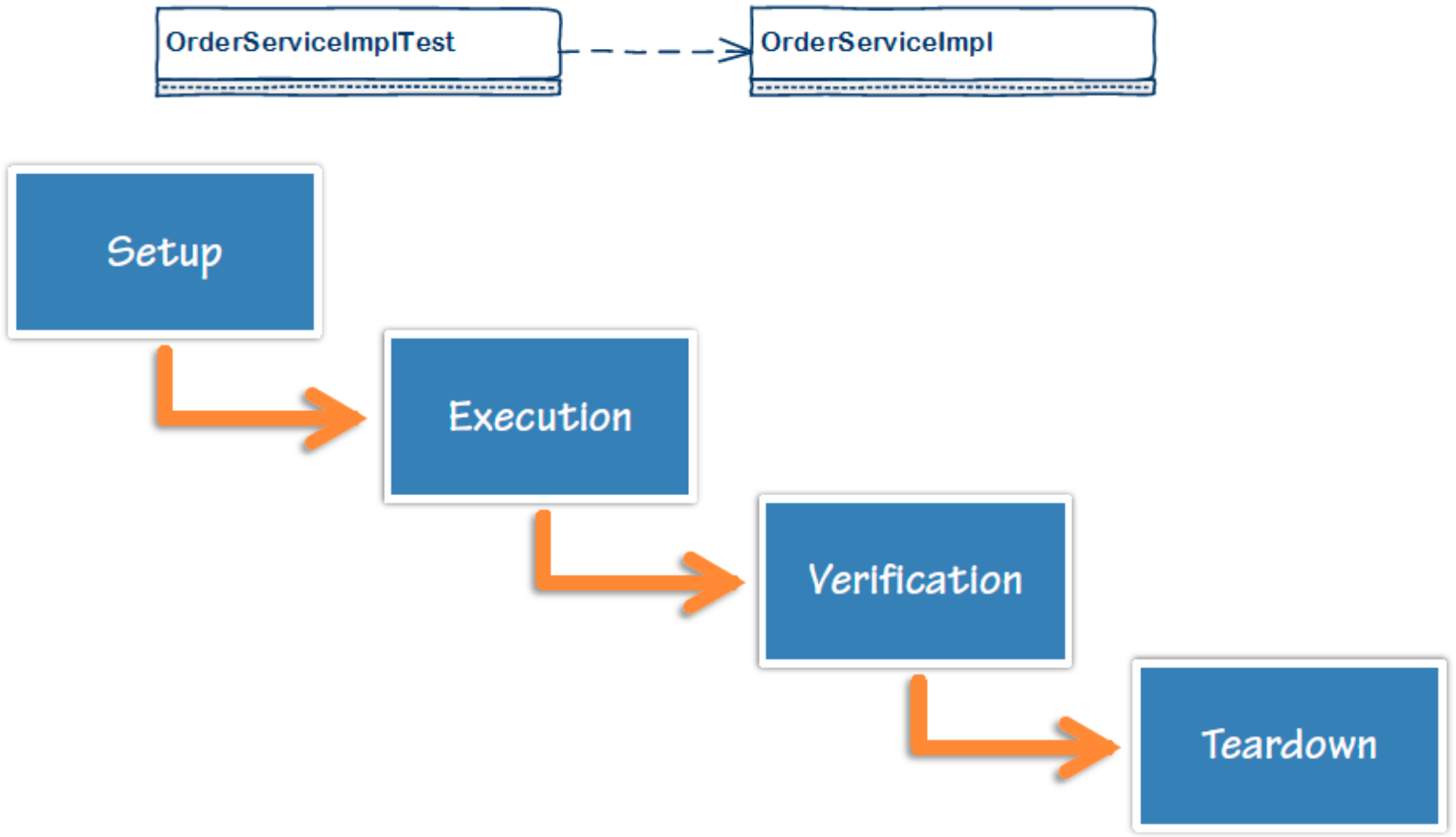
Junit

Outline



1. Overview
2. Junit Core Construct
3. Demo: Red / Green / Refactor
4. Summary

Overview of Junit Testing Concepts



Junit Core Construct

All annotations listed are in the 'org.junit' package

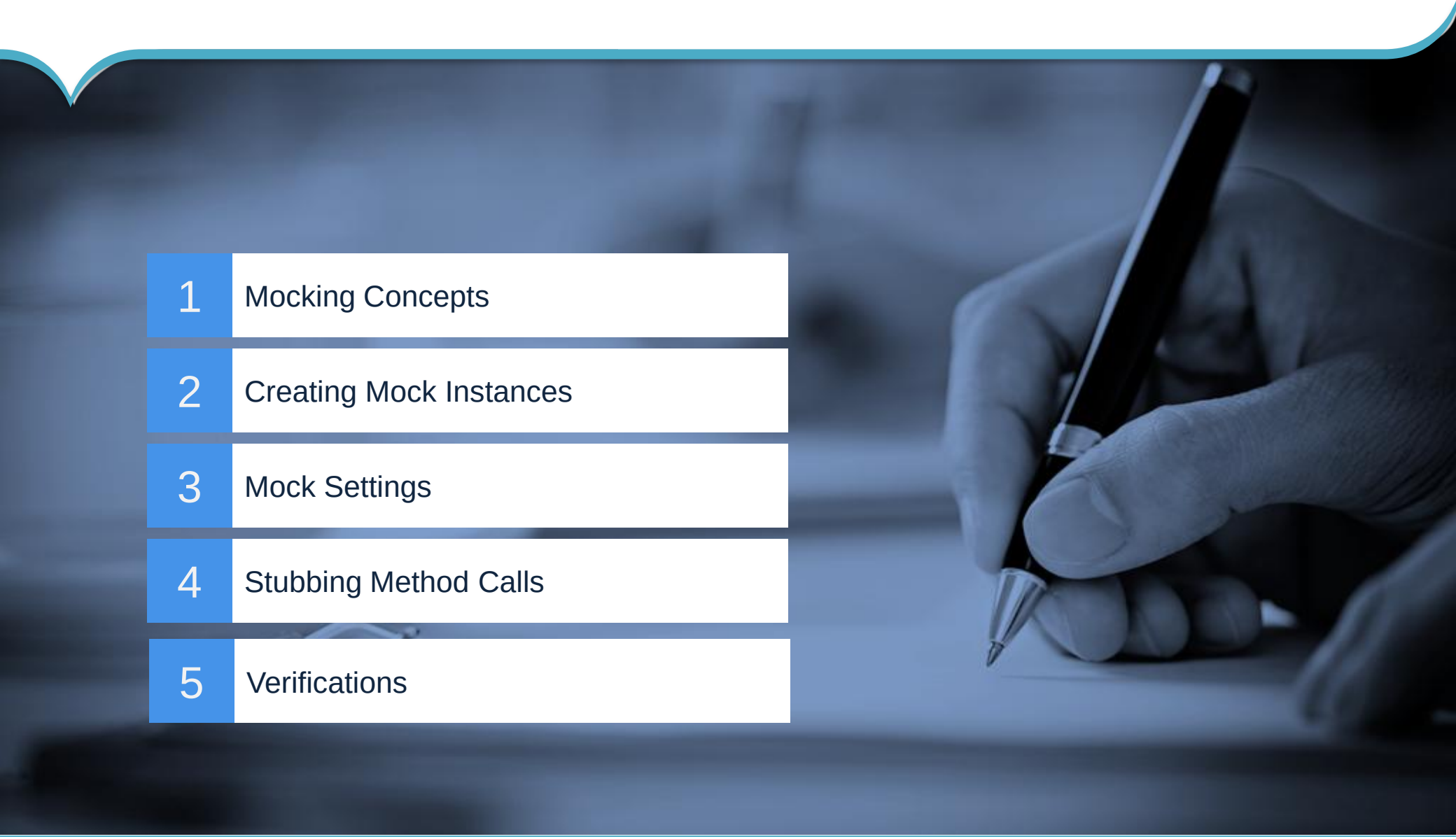
- **@Test**
- **Verification Methods:**
  - **org.junit.Assert.***
  - **HamcrestMatcherAssert**
- **@Before / @BeforeClass**
- **@After / @AfterClass**

Summary

- ❑Basic elements of an automated unit test
- ❑Core features of the JUnit framework
- ❑Detailed demonstration of Red / Green / Refactor

Mockito

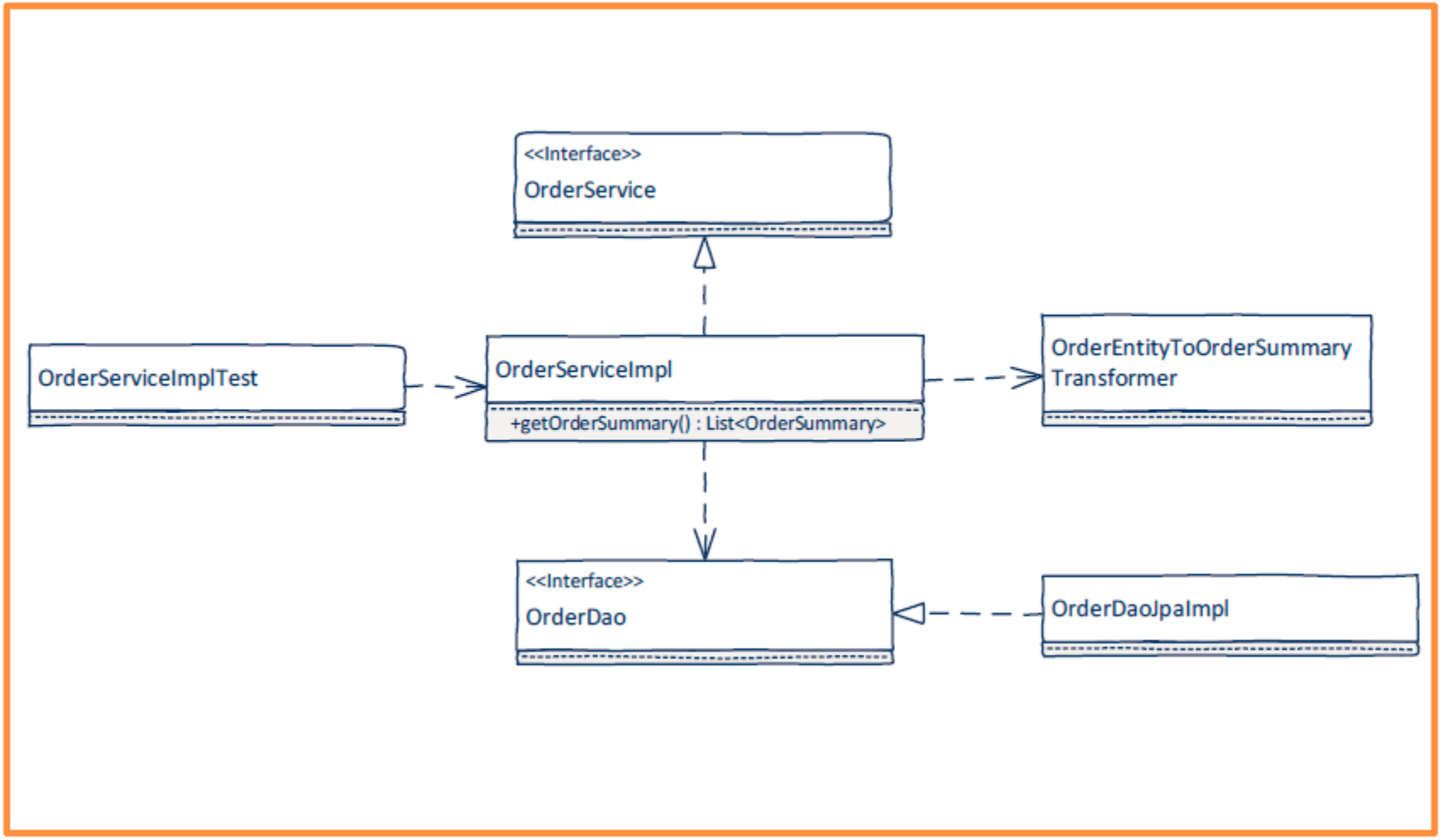
Outline



1. Mocking Concepts
2. Creating Mock Instances
3. Mock Settings
4. Stubbing Method Calls
5. Verifications

Mocking Concepts

Mocking Concepts

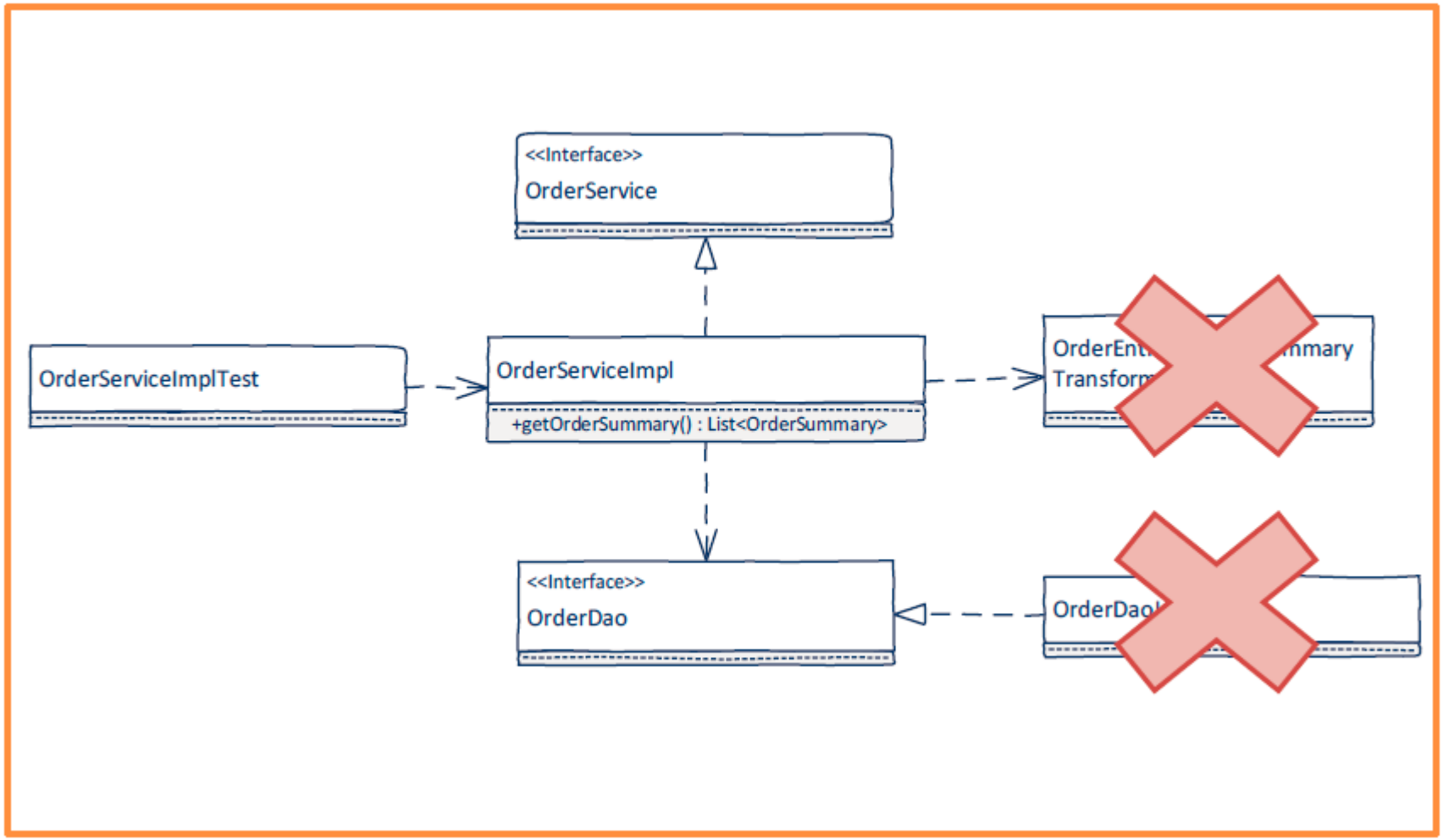


Mocking Concepts

- **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)*

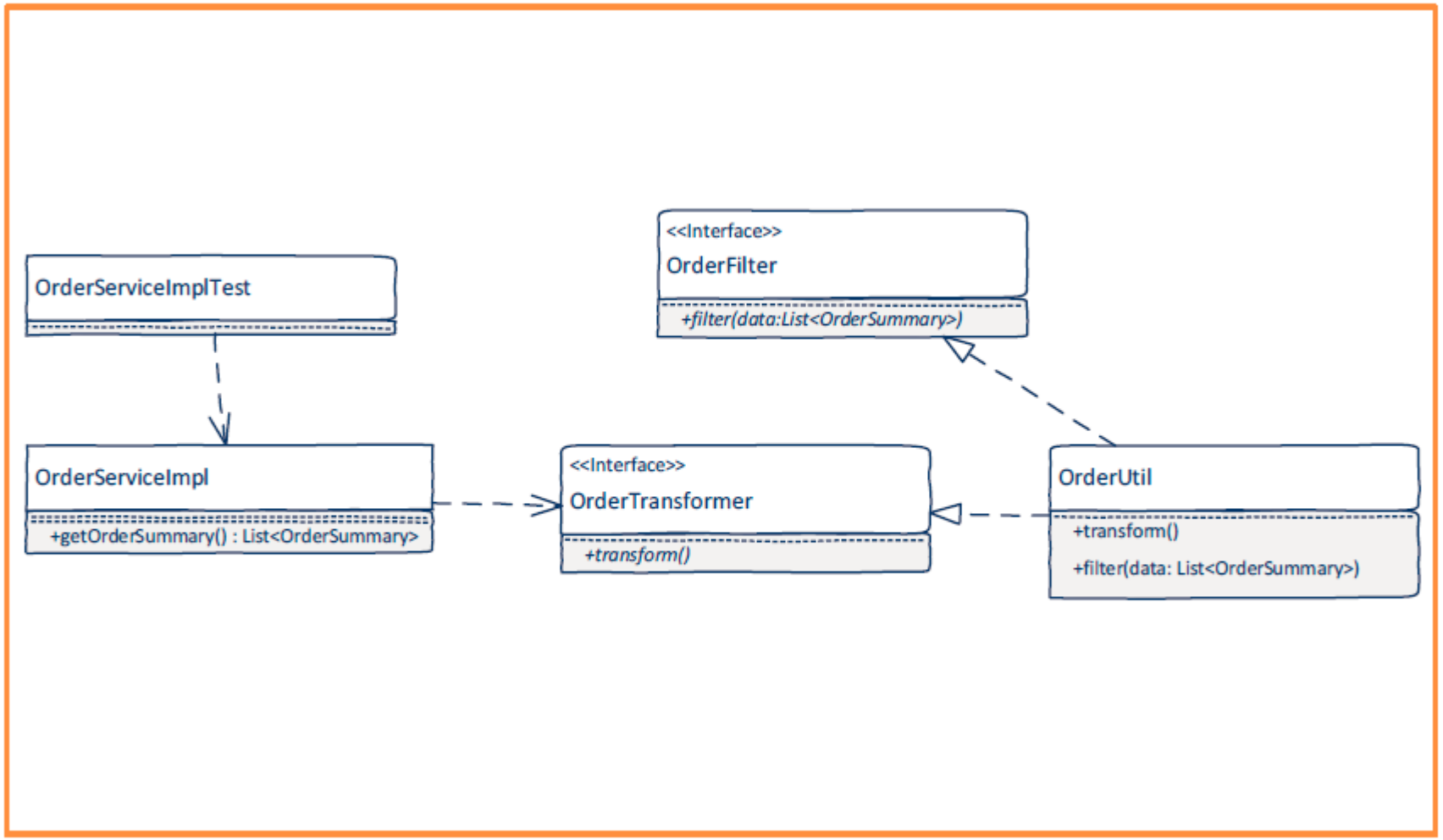
Creating Mock Instances

- **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

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

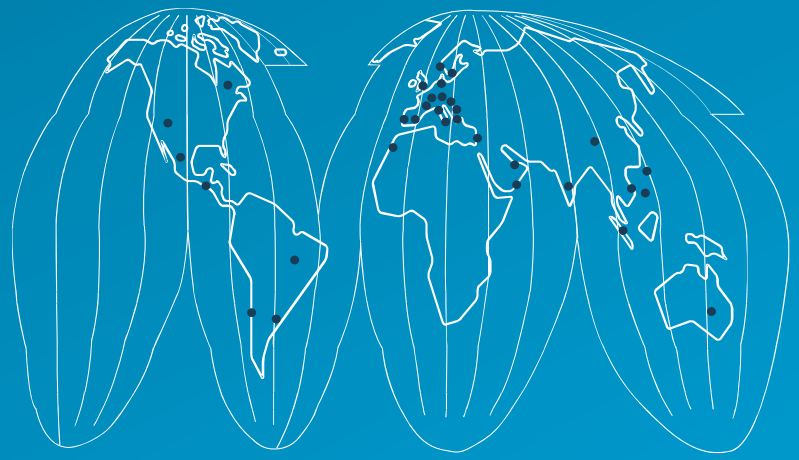
People matter, results count.

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