DEPENDENCIES

PBA SOFTWAREUDVIKLING/
BSC SOFTWARE DEVELOPMENT

Christian Nielsen cnls@cphbusiness.dk

Tine Marbjerg tm@cphbusiness.dk

SPRING 2019

TODAY'S TOPICS

- **Overview**
  - Assignments / PeerReviews / Exam
  - Learning objectives
  - Basic Test Project
  - Test Suites
  - Dependency injection / Inversion of control / Interfaces
  - Test doubles: Mocks / Stubs / Fakes / Dummies / Spies
  - State / Behaviour
  - Mockito
  - Examples
  - Exercises
  - Assignment

LEARNING OBJECTIVES

- Use interfaces and apply dependency injection to make code more testable
- Know the difference between mocks, stubs, fakes, spies and dummies and when to use them
- Perform state testing and behavior testing
- Be able to setup and use Mockito

EXERCISES

- BASIC TEST PROJECT
- TEST SUITES
- CALENDAR MOCKING
- ORDER STUBBING AND MOCKING
- OWN EXAMPLE

BASIC TEST PROJECT

Dependencies - Basic Test Project (Netbeans Maven)

org.junit.jupiter: junit-jupiter-engine 5.4.0

org.junit.platform: junit-platform-runner 1.4.0

org.junit.Jupiter: junit-jupiter-params 5.4.0

org.hamcrest: hamcrest-core 2.1

org.hamcrest: hamcrest-library 2.1

org.mockito: mockito-core 2.24.0

TEST SUITES

EXAMPLE: JUnit4 Test Suite

```
package junit4;


import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;


@RunWith(Suite.class)
@SuiteClasses({T_E_S_T_JUnit4.class})
public class T_E_S_T_JUnit4Suite_SuiteClass {}
```

TEST SUITES

EXAMPLE: JUnit5 Test Suite

```
package junit5;

import org.junit.runner.RunWith;
import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.platform.suite.api.IncludeClassNamePatterns;

@RunWith(JUnitPlatform.class)
@SelectPackages("hamcrest")
@SelectClasses({TestSimpleJUnit5.class,T_E_S_T_JUnit5.class})
@IncludeClassNamePatterns({"^.*Test.*|^.*T_E_S_T_.*"})
public class T_E_S_T_JUnit5Suite_JUnitPlatformClass {}
```

TEST SUITES

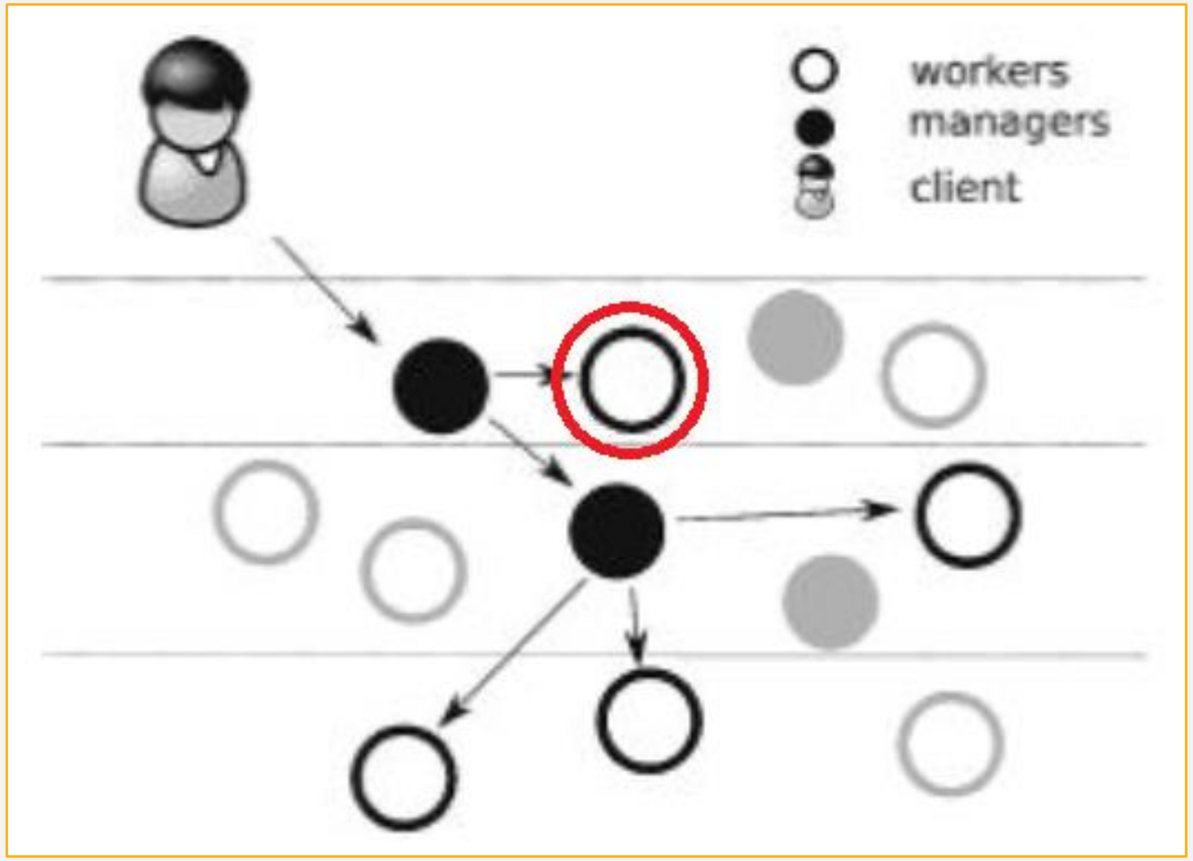
EXAMPLE: JUnit4 Test File

```
public class T_E_S_T_JUnit4
{
    @Test
    public void testSimple()
    {
```

EXAMPLE: JUnit5 Test File…

```
@RunWith(JUnitPlatform.class)
public class T_E_S_T_JUnit5
{
    @Test
    void testSimple()
    {
```

TESTS WITH DEPENDENCIES



TESTS WITH DEPENDENCIES

Code isolation…

One piece of code knows little or nothing about other pieces of code

Dependencies…

Required objects, components, resources needed by a given piece of code

Dependency injection…

The process of supplying objects, components, resources that a given piece of code requires

Different strategies: Constructor injection / Setter injection / Interface injection / Framework injection

Inversion of control…

Client has control over which implementation to use by injecting the dependencies

Interfaces…

A reference type / A collection of abstract methods / Interface contract for implementations / Makes multiple implementations possible

TESTS WITH DEPENDENCIES

What if test unit depends on classes / systems that…

Is not yet created?

Provide behavior not acceptable for a unit test (prints, send a mail, controls external hardware etc.)?

Relies on a database (takes a long time to start, on a remote server, must be kept clean etc.)?

Is complex and itself relies on external resources (web-service calls, file-I/O, external hardware etc.)?

Supplies non-deterministic results (current time/date, temperature etc.)?

TEST DOUBLES

Surround objects under test with predictable test doubles

Test doubles are used to replace dependencies making it possible to:

- Gain full control over the environment in which the test unit is running
- Verify interactions between the test unit and it’s dependencies

Test doubles can be used to both mock dependencies and verify how methods in dependencies are called and interact

TEST DOUBLES

Mocks / Mock objects

Pre-programmed objects with expectations, which form a specification of the calls they are expected to receive

Fakes / Fake objects

Have simplified working implementations of production code

Stubs / Stub objects

Hold predefined data used to answer methods calls

Spies / Spy objects

Spies are stubs that also record some information based on how they were called

Dummies / Dummy Objects

Objects passed around but never used

TEST VERIFICATION

There is a difference in how test results are verified: a distinction between state verification and behavior verification

State Verification / State testing

Object under testing perform a certain operation, after being supplied with all necessary dependencies

Verify ending state of the object and/or the dependencies is as expected

Behaviour Verification / Behaviour testing

Specify exactly which methods are to be invoked on the dependencies by the test unit

Verify that the sequence of steps performed was correct

MOCKING FRAMEWORKS

Many different mocking frameworks exist

Java Mocking Frameworks: JMock / EasyMock / JMockit / **Mockito**

MOCKITO

Voted the best mocking framework for java

Top 10 Java library across all libraries, not only the testing tools

MOCKITO

MOCKITO PHASES

Mockito has essentially two phases, one or both of which are executed as part of unit tests, stubbing and verification

Stubbing

Stubbing is the process of specifying the behavior of mocks

Specify what should happen when interacting with mocks

Make it possible to control the responses of method calls in mocks, including forcing them to return any specific values or throw any specific exceptions

Verification

Verification is the process of verifying interactions with mocks

Determine how mocks were called and how many times

Look at the arguments of mocks to make sure they are as expected

MOCKITO

MOCKITO TEST DOUBLES

Mock

A complete mock or fake object mock is created, where the default behavior of the methods is do nothing, which can then be changed

With a Mock instance both state and behavior can be tested

A mock is not created from an actual instance

Instrumented to set up expectations and track interactions

Spy

Spies should be used carefully and occasionally, such as when dealing with legacy code

Spy is created from an actual instance will wrap an existing instance

Calls the real implementation of the methods

Instrumented to track interactions

MOCKITO

MOCKITO BASICS

```
//Init mocks
MockitoAnnotations.initMocks(this);

//Create mocks
MyClass mc = mock(MyClass.class);

//Specify behavior of mocks
when(mc.myMethod(10)).thenReturn("Hello");

//Verify mocks
verify(mc, times(1)).myMethod(10);
```

RESOURCES…

Dependencies

https://www.danclarke.com/writing-testable-code-its-all-about-dependencies

Test doubles

http://xunitpatterns.com/Mocks,%20Fakes,%20Stubs%20and%20Dummies.html

https://martinfowler.com/articles/mocksArentStubs.html

Mockito Documentation

https://static.javadoc.io/org.mockito/mockito-core/2.24.5/index.html?org/mockito/Mockito.html

Baeldung Mockito

https://www.baeldung.com/tag/mockito/

https://www.baeldung.com/mockito-series

JavaCodeGeeks Mockito

https://www.javacodegeeks.com/2015/11/testing-with-mockito.html

DZone Mockito

https://dzone.com/refcardz/mockito

Vogella Mockito

https://www.vogella.com/tutorials/Mockito/article.html

Tutorialspoint Mockito

https://www.tutorialspoint.com/mockito/