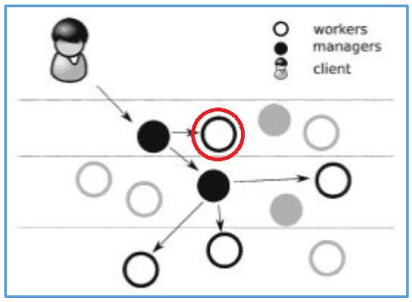
# DEPENDENCIES

## 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

## TESTS WITH DEPENDENCIES

Typically in an object oriented design a single task is solved by delegating subtasks to a number of objects by passing messages between them and between layers and tiers




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
Providing objects, components, resources that an object needs (its dependencies) instead of having it construct them itself
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

Dependency injection, inversion of control and interfaces helps in designing loosely coupled classes, which make them testable, maintainable and extensible

When writing a class…

Remove internal and external dependencies

Do not initialize dependencies in classes

When testing a class…

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.)?

SUT: System under test

The part of the system being tested.

Depending on the type of test the granularity can range from a single class to the complete system

Unit to test

DOC: Depend on component

Any entity that is required by the SUT to fulfill its duties

Dependencies

## TEST DOUBLES

Clean code is code that does one thing well

Clean tests are focused tests that fail for only one reason

How to get clean code and clean tests -> Surround objects under test with predictable test doubles

Test doubles are used to replace DOC's allowing us to:

- Gain full control over the environment in which the SUT is running
- Verify interactions between the SUT and its DOC's

It is very easy to get confused about the terminology related to test doubles

Many articles use different terms to mean the same thing, and sometimes even mean different things for the same term

The most widely used terms related to test doubles: Mocks, Stubs, Fakes, Dummies, Spies

**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

When using test doubles in tests, test doubles are first set up with behavior before injection and then method calls are registered, verified and asserted

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

## 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 SUT
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 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
Stubbing makes it simple to create all the possible conditions for tests
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
Code different behaviors under different conditions and control exactly what mocks should do

**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
Ensure that exactly the expected values were passed to dependencies and that nothing unexpected happens
Determine exactly what happened to mocks

### 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
Creates a bare-bones shell instance
Instrumented to set up expectations and track interactions

**Spy**
A real object where it is possible to track specific methods of it
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 MOCKING

**Initialize mocks**
*Alternative 1*
MockitoAnnotations.initMocks(this);

*Alternative2*
@BeforeEach
public void setUp() { MockitoAnnotations.initMocks(this); }

**Create mocks**
*Alternative 1*
MyClass mc = mock(MyClass.class);

*Alternative 2*
@Mock
MyClass mc = new MyClass();

**Create spies**
*Alternative 1*
MyClass mc = spy(MyClass.class);

*Alternative 2*
@Spy
MyClass mc = new MyClass();

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

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

**Capture arguments of mocks**
ArgumentCaptor<String> argumentCaptor = ArgumentCaptor.forClass(String.class);
verify(mockedList, times(3)).add(argumentCaptor.capture());
assertThat(List.of("a1", "b2", "c3"), is(argumentCaptor.getAllValues()));

## 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/