**Testing with EasyMock - Tutorial**

Table of Contents

* [1. Prerequisites](http://www.vogella.com/tutorials/EasyMock/article.html#prerequisites)
* [2. EasyMock](http://www.vogella.com/tutorials/EasyMock/article.html#easymock_overview)
* [3. Download Easy Mock](http://www.vogella.com/tutorials/EasyMock/article.html#easymock_installation)
* [4. Tutorial: Using Easy Mock and JUnit](http://www.vogella.com/tutorials/EasyMock/article.html#tutorial_easymock)
* [5. Links and Literature](http://www.vogella.com/tutorials/EasyMock/article.html#resources)
* [6. vogella training and consulting support](http://www.vogella.com/tutorials/EasyMock/article.html#vogella-training-and-consulting-support)
* [Appendix A: Copyright, License and Source code](http://www.vogella.com/tutorials/EasyMock/article.html#copyright-license-and-source-code)

Lars Vogel, (c) 2011, 2016 vogella GmbH Version 1.5, 01.09.2016

*Testing with EasyMock. This tutorial explains testing with the EasyMock framework within Eclipse. It is based on the EasyMock 3.1 release.*

[**1. Prerequisites**](http://www.vogella.com/tutorials/EasyMock/article.html#prerequisites)

The following tutorial is based on an understanding of unit testing with the JUnit framework.

In case your are not familiar with JUnit please check the following [JUnit Tutorial](http://www.vogella.com/tutorials/JUnit/article.html).

[**2. EasyMock**](http://www.vogella.com/tutorials/EasyMock/article.html#easymock_overview)

*EasyMock* is a mock framework which can be easily used in conjunction with JUnit. The following description demonstrates the usage of EasyMock.

EasyMock instantiates an object based on an interface or class.

**import** static org.easymock.EasyMock.createNiceMock;

....

*// ICalcMethod is the object which is mocked*

ICalcMethod calcMethod = createNiceMock(ICalcMethod.class);

The createNiceMock() method creates a mock which returns default values for methods which are not overiden. A mock created with the Mock() method will fails in such a case.

EasyMock has several methods which are used to configure the Mock object. Theexpect() method tells EasyMock to simulate a method with certain arguments. TheandReturn() method defines the return value of this method for the specified method parameters. The times() method defines how often the Mock object will be called.

The replay() method is called to make the Mock object available.

*// setup the mock object*

expect(calcMethod.calc(Position.BOSS)).andReturn(70000.0).times(2);

expect(calcMethod.calc(Position.PROGRAMMER)).andReturn(50000.0);

*// Setup is finished need to activate the mock*

replay(calcMethod);

[**3. Download Easy Mock**](http://www.vogella.com/tutorials/EasyMock/article.html#easymock_installation)

Download EasyMock from the [EasyMock Homepage](http://easymock.org/) and add the easymock.jar library to your classpath.

You also need to download the [Objenesis](http://objenesis.org/download.html) and [Cglib](http://cglib.sourceforge.net/) libraries and add these jars to your classpath.

[**4. Tutorial: Using Easy Mock and JUnit**](http://www.vogella.com/tutorials/EasyMock/article.html#tutorial_easymock)

[**4.1. Create project and classes**](http://www.vogella.com/tutorials/EasyMock/article.html#tutorial_easymockproject)

Create a new Java Project called *com.vogella.testing.easymock.first*. Create the following classes.

**package** com.vogella.testing.easymock.first;

**public** **enum** Position {

BOSS, PROGRAMMER, SURFER

}

**package** com.vogella.testing.easymock.first;

**public** **interface** **ICalcMethod** {

**double** calc(Position position);

}

**package** com.vogella.testing.easymock.first;

**public** **class** **IncomeCalculator** {

**private** ICalcMethod calcMethod;

**private** Position position;

**public** **void** setCalcMethod(ICalcMethod calcMethod) {

this.calcMethod = calcMethod;

}

**public** **void** setPosition(Position position) {

this.position = position;

}

**public** **double** calc() {

**if** (calcMethod == null) {

**throw** **new** RuntimeException("CalcMethod not yet maintained");

}

**if** (position == null) {

**throw** **new** RuntimeException("Position not yet maintained");

}

**return** calcMethod.calc(position);

}

}

The IncomeCalculator class should be tested. The class has the purpose to calculate the salary of a person based on the provided method and position. Obviously the test depends on the provided methods.

[**4.2. Create tests**](http://www.vogella.com/tutorials/EasyMock/article.html#tutorial_easymocktest)

Create a new test source folder in your project.

Create a new test for IncomeCalculator and place the new test class in this folder.

**package** com.vogella.testing.easymock.first.test;

*// use static imports to*

*// have direct access to these methods*

**import** static org.easymock.EasyMock.createNiceMock;

**import** static org.easymock.EasyMock.expect;

**import** static org.easymock.EasyMock.replay;

**import** static org.easymock.EasyMock.verify;

**import** static org.junit.Assert.assertEquals;

**import** static org.junit.Assert.fail;

**import** org.junit.Before;

**import** org.junit.Test;

**import** com.vogella.testing.easymock.first.ICalcMethod;

**import** com.vogella.testing.easymock.first.IncomeCalculator;

**import** com.vogella.testing.easymock.first.Position;

**public** **class** **IncomeCalculatorTest** {

**private** ICalcMethod calcMethod;

**private** IncomeCalculator calc;

@Before

**public** **void** setUp() **throws** Exception {

*// NiceMocks return default values for*

*// unimplemented methods*

calcMethod = createNiceMock(ICalcMethod.class);

calc = **new** IncomeCalculator();

}

@Test

**public** **void** testCalc1() {

*// Setting up the expected value of the method call calc*

expect(calcMethod.calc(Position.BOSS)).andReturn(70000.0).times(2);

expect(calcMethod.calc(Position.PROGRAMMER)).andReturn(50000.0);

*// Setup is finished need to activate the mock*

replay(calcMethod);

calc.setCalcMethod(calcMethod);

**try** {

calc.calc();

fail("Exception did not occur");

} **catch** (RuntimeException e) {

}

calc.setPosition(Position.BOSS);

assertEquals(70000.0, calc.calc(), 0);

assertEquals(70000.0, calc.calc(), 0);

calc.setPosition(Position.PROGRAMMER);

assertEquals(50000.0, calc.calc(), 0);

calc.setPosition(Position.SURFER);

verify(calcMethod);

}

@Test(expected = RuntimeException.class)

**public** **void** testNoCalc() {

calc.setPosition(Position.SURFER);

calc.calc();

}

@Test(expected = RuntimeException.class)

**public** **void** testNoPosition() {

expect(calcMethod.calc(Position.BOSS)).andReturn(70000.0);

replay(calcMethod);

calc.setCalcMethod(calcMethod);

calc.calc();

}

@Test(expected = RuntimeException.class)

**public** **void** testCalc2() {

*// Setting up the expected value of the method call calc*

expect(calcMethod.calc(Position.SURFER)).andThrow(

**new** RuntimeException("Don't know this guy")).times(1);

*// Setup is finished need to activate the mock*

replay(calcMethod);

calc.setPosition(Position.SURFER);

calc.setCalcMethod(calcMethod);

calc.calc();

}

}

After execution of the test you can call the verify method to check if the mock object was called as defined.