# Unit Testing

(individual work, 2 weeks)

# Introduction

In this assignment, you will work with the JUnit framework to perform Java unit testing. JUnit is an open source project available at [www.junit.org](http://www.junit.org) which is already integrated into Eclipse 3.4 and later. We will be working with JUnit version 4. This Lab consists of two parts. In the first part you will learn a bit about JUnit by doing a few tutorial-like mini-exercises. In the second section you work on a set of tasks with the classes Currency, Money and Bank.

## Basic Information

Before you begin, we would like to call your attention to the following conventions:

A Test Case Class is named [classname]Test.java, where classname is the name of the class that is tested.

A Test Case Method is a method of the Test Case Class which is used to test one or more of the methods of the target class. Test Case Methods are annotated with @Test to indicate them to JUnit. Cases without @Test will not be noticed by JUnit.

JUnit assertions are used to assert that a condition must be true at some point in the test method. JUnit has many types of assertions. The following is a selection of the most commonly used assertions:

* assertEquals(expected, actual): Assert that expected value is equal to the actual value. The expected and actual value can be of any type, for example integer, double, byte, string, char or any Java object. If the expected and actual values are of type double or float, you should add a third parameter indicating the delta. It represents the maximum difference between expected and actual value for which both numbers are still considered equal.
* assertTrue(condition): Asserts that the Boolean condition is True.
* assertFalse(condition): Asserts that the Boolean condition is False.
* assertNull(object): Asserts that an object is null.
* assertNotNull(object): Asserts that an object is not null.
* assertSame(expected object, actual object): Asserts that two variables refer to the same object.
* assertNotSame(expected object, actual object): Asserts that two variables do not refer to the same object.

Whenever an assertion fails, an AssertionError is thrown, which is caught by the JUnit framework and presented as a red bar, indicating test failure. Assert statements accept an extra message parameter before the other parameters. This parameter is a String which will be displayed if the assertion fails.

Exceptions will also be caught by JUnit, and cause the test to fail. Except when you indicate that it is expected in the annotation, like this:@Test(expected=SomeException.class).If you indicate an expected exception, the test will fail if that exception is not thrown.

## Loading the Project

Download the Lab1. file

In Eclipse, choose File -> New -> Java Project. Give it a name (\Lab1", for instance) and click Finish.

To load Lab1.zip into this project, choose File -> Import -> Archive File, Browse for Lab1.zip, click Finish and answer Yes to letting it overwrite the .classpath and .project files (this is important, otherwise you will have to mess with setting up the classpath and source libraries yourself).

You should now have a project with a test/and a src/directory, and there should be a number of errors showing up from the Money and Currency classes in src/b\_Money.

This is supposed to happen, and if Eclipse complains about it during the tutorial exercises, you may ignore it until you get to the Money part of the Lab.

## Tutorial Exercises

The package a\_Introductory contains classes in both the test/and the src/library.

The unit test classes are to be found in test/a\_Introductory/and the classes to be tested are in src/a\_Introductory.

### Fibonacci

As the first mini-exercise, we take a look at the Fibonacci class. This class is an attempt at implementing the recursive method fib, which should generate the nth Fibonacci Number.

Notice that in FibonacciTest.java, there is a testFib method corresponding to the fib method in the Fibonacci class. If you ever wish to create test classes, Eclipse has a wizard which will create these corresponding test methods for you.

In the test class we assert that the first 7 Fibonacci numbers must be equal to 0,1,1,2,3,5,8,13 by writing the expected value and calling the fib method to generate the number. If the fib fails at generating the right number, it will throw an AssertionError with a message showing which number it failed to generate, and the JUnit framework will show the infamous red bar of a failed test.

To read about Fibonacci numbers see http://en.wikipedia.org/wiki/Fibonaccinumber

## Running the Tes**t**Cases

When you run a test class, JUnit will run each method annotated with @Test separately and show a green bar if all of them pass, and a red bar if any of them fail. It is important that anything happening in a test method is independent from the other test methods, otherwise you risk getting weird results. But in the case of Fibonacci, we have only one test method.

Do the following:

1. To run the test, select the FibonacciTest.java in the project explorer and choose Run -> Run As -> JUnit Test. Notice the new JUnit window appears and shows a red bar indicating that the test failed. Below the listing of the running test cases you can see a stack of all failed test cases and their messages.

2. Now, find the bug in Fibonacci.java, correct it and run the test again. Note with satisfaction the green bar and move on to the Quadrilateral exercise.

### Quadrilateral

The Quadrilateral class defines a polygon with four sides. It has two methods: isRectangle and isSquare. Furthermore, it also uses the classes Point, Line and Vector2D.

To find if the polygon is a rectangle, we use vectors and dot products to determine if every corner forms a right angle. To find if the polygon is a square, we use isRectangle and check if the lengths of all sides are equal.

Do the following:

1. Look through the test classes for Quadrilateral, Point, Line and Vector2D and read the source code for the tests.

2. Run the test cases and note that not all of them succeed. The test cases are correct, but the code has been littered with a number of bugs. Examine which test cases that failed and check the failure messages. Find and correct the bugs in the code.

## Assignment Tasks

### Money and Currency

You have been given a template for the Currency and Money classes, with JavaDoc comments explaining what each method should do. There are also classes Bank and Account, but we will come back to that in the following section. All the methods of Currency.java and Money.java are empty.

First, write test cases for the methods of each class, and then fill in the methods with code that will make your test cases pass. The template test methods are a guideline you can use for constructing your tests. Unless you have a good idea for how to otherwise structure the tests, you should simply fill in those template test methods.

Important note: The decimal numbers representing money are implemented as integers. The last two digits denote two decimals. So Money(200050, SEK) will mean 2000.50 SEK.

### Bank

The Bank and Account classes were written by a bad programmer. When you are confident your Money and Currency classes work as intended, write test cases for the Bank and Account classes and find the bugs. Improve the code if you are able to. Again, the specification is provided in the JavaDoc comments.

### To Do Checklist

1. Write the code for the body of the test case methods of MoneyTest.java and CurrencyTest.java.

2. Motivate your test cases by commenting in the test methods.

3. Complete the methods body in Money.java and Currency.java classes, and comment where necessary. Again make sure you follow the specification for each method and implement the required functionality.

4. If you run your test cases for MoneyTest.java and CurrencyTest.java at this point, they should be all pass.

5. By following the Bank.java and Account.java specification in the JavaDoc comments, write your test cases in theBankTest.java and AccountTest.java files. Note which of your test cases fail, by commenting in the corresponding test methods.

6. Find the bugs inBank.javaandAccount.java based on the JUnit failures from the previous step, and note it by placing a comment in the code where the bug was spotted. Explain how you found the bug with your unit tests.

7. Fix the bugs and verify that BankTest.java and AccountTest.java passes successfully.