Lab Report for SQAT

|  |  |
| --- | --- |
| **Student ID:** | L201726630108 |
| **Student Name:** | Tamraoui Oussama |

.

|  |
| --- |
| 1. **Rename. Please rename this file (Labwork.docx) to “Your Name- Student ID.docx”, e.g.,** Tamraoui Oussama**-** L201726630108**.docx.** 2. **Submission.**    1. **Please submit your Lab Report as well as the corresponding Source Codes to Chaoxing System.**    2. **Note that although you have several assignments, you only submit one Lab Report (.docx) and one source code package (.zip) which are organized as follows:**      1. **Attention:**    1. **You should report the problem you met and the solving methods you got during the whole procedure.**    2. **You should snapshot the important result of your running results.**    3. **You do not need to paste your code into the lab report unless it’s really necessary to illustrate your answer.** |

# Lab01: JUnit 4 for Unit Test

**Deadline:** 03/19/2020, 12 p.m.

## Target

* To be familiar with the IDE: [Eclipse](https://www.eclipse.org/) / [IntelliJ IDEA](file:///E:\Work\Job\Teaching\SoftwareTest(overseas)\Experiments\Lab1\IntelliJ%20IDEA)
* To understand Java's **annotations**
* To understand the basic concept of **Unit Test**
* To be familiar with Unit Test skills with [**JUnit**](https://junit.org/junit4/) **4**:
* Assert Functions, e.g., assertTrue, assertFalse.
* JUnit 4.0 Configuration in your IDE(Integrated Development Environment), e.g., [Eclipse](https://www.eclipse.org/), [IntelliJ IDEA](file:///E:\Work\Job\Teaching\SoftwareTest(overseas)\TestArt\src\lab01\IntelliJ%20IDEA) (Although Eclipse is very popular, I ***strongly*** suggested you to use IDEA. It’s really ***Excellent*** & ***Fascinating***!).

## Tools

* IDE: [Eclipse](https://www.eclipse.org/) / [IntelliJ IDEA](file:///E:\Work\Job\Teaching\SoftwareTest(overseas)\Experiments\Lab1\IntelliJ%20IDEA) / any IDE you’d like to use
* Programming Language: Java

## Tasks

### Task 01: Terminology Illustration

Please illustrate the following Terminology about Unit Test:

* + - 1. What’s test suite?
      2. What’s test case?
      3. What’s unit test?
      4. What’s test fixture?
      5. **Annotations**. JUnit 4.0 uses **annotations** rather than special names for setting up, tearing down and testing. Please list the mainly used **annotations** in JUnit 4.0.
      6. **Assert** is very important for Making Tests. Please read and run the test codes, CalculatorTest and TriangleTest, in *LectureCodes/src/lec01*, and illustrate why? That is, if we don’t use Assert functions, what will happen in our testing code?
      7. **Assert Statement**. There are two forms of the assert statement. Please illustrate what they are.

### Task 02: Set up JUnit in Eclipse / IDEA

Set up JUnit 4.0 in Eclipse / IDEA, and record your setting procedure by text description and screen snapshot. Please check your setting by running the test codes, CalculatorTest and TriangleTest in LectureCodes/src/lec01.

### Task 03: Build your first JUnit testing function

Please add the following method “int sub(int i, int j)” into the class [Cal](Lab01/src/task03/Cal.java), and then add a new testing method in [CalTest](Lab01/src/task03/CalTest.java) class by referring to its testAdd method.

public int sub(int i, int j) {

return i - j;

}

### Task 04: Test your own code with JUnit 4.0

Basic Employee Compensation Problem. For each week, hourly employees are paid a standard wage per hour for the first 40 hours worked, 1.5 times their wage for each hour after the first 40 hours, and 2 times their wage for each hour worked on Sundays and Holidays. Table 1 gives some test cases of this.

Please write a Java class, WageCalculator, to solve the wage problem in the following and a test class WageCalculatorTest to test your code by using the test cases in Table 1.

Table 1 Test Cases for Basic Employee Compensation Problem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Expected** | **Testing Result** |
| StandardHours | HolidayHours | HourlyWage |
| 40 | 0 | $20 | $800 | Pass |
| 45 | 0 | $20 | $950 | Pass |
| 48 | 8 | $20 | $1280 | **Fail**  Actual: $1360 |

# Lab02: JUnit 5 for Unit Test

**Deadline:** 03/29/2020, 12 p.m.

**Note: in this labwork, all of the related codes are in Labwork/Lab02/src.**

## Target

* To know about **Maven** and Maven-based Project
* To understand how to **add dependencies** by Maven
* To understand **Parameterized Test**
* To understand **Timeout Testing** and **Exception Test**

## Tools

* IDE: [Eclipse](https://www.eclipse.org/) / [IntelliJ IDEA](file:///E:\Work\Job\Teaching\SoftwareTest(overseas)\Experiments\Lab1\IntelliJ%20IDEA) / any IDE you’d like to use
* Programming Language: Java

## Tasks

### Task 01: Config JUnit 5 in a Maven Project

All things are difficult before they become easy! In this subtask, you almost do nothing except for configuring JUnit 5. However, it might be a little challenge! I spent a lot of time!

IntelliJ IDEA supports running tests on the JUnit Platform since version 2016.2. However, only IDEA 2017.3 or newer versions of IDEA will download the following JARs automatically based on the API version used in the project: *junit-platform-launcher*, *junit-jupiter-engine*, and *junit-vintage-engine*. In addition, [Parameterized Tests](https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests), which are very important in JUnit testing, are not automatically supported by any version of IDEA. You need to add a *dependency* on the *junit-jupiter-params* artifact.

In this subtask, I provided you a *well-configured* Maven project, which added all of the JUnit 5 *dependencies* we’ll use in the following tasks. Please do the following subtasks to test if your JUnit 5 packages are well set in your IDE. In the following, I suppose you use IDEA, instead of Eclipse.

Find the ***settings.xml*** of your Maven, which lies in your Maven install path,like %IDEA\_INSTALL\_PATH%/plugins/maven/lib/maven3/conf/settings.xml. **Revise <mirrors> as follows**:

<mirrors>

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>\*</mirrorOf>

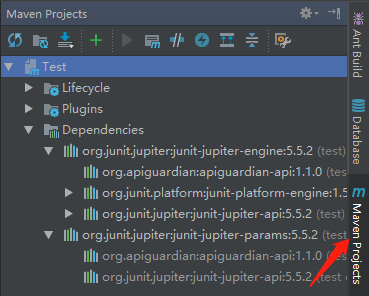
<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

</mirrors>

Restart IDEA and check if the JUnit 5 dependencies in your Maven project are well-installed. If well-installed, there will be no any error tips as follows:



To test if your JUnit 5 can well work, please run the test method, testAppleColor, in *task01.AppleTest*. If you get the following result, congratulations, you make it!



Now, please illustrate where we add the JUnit 5 dependencies in this project? Why we manage these dependencies by maven, rather than manually? Please give your answer in detail.

Please create a new Maven project as follows and add the JUnit 5 dependencies where you found in the previous step and copy them to your new project. And also, copy the two classes, Apple and AppleTest, to your new project. Finally, try to run testAppleColor in AppleTest again and snapshot the result.

### Task 02: Timeout Testing and Exception Test

Both *Timeout Testing* and *Exception Test* are very important. The class, *task02.Calculator*, has two methods, *squareRoot* and *divide*. In the following, we have 2 subtasks to perform *Timeout Testing* and *Exception Test* on the above 2 methods. Please do them carefully.

***Timeout Testing***. In 2 scenarios, we need to use Timeout Testing. ***First***, it can be used for ***performance test*** to ensure the method is returned within a reasonable time. ***Second***, it is advisable to use Timeout Testing when we write test cases with external dependencies, which never give a 100% certainty that they will be available while executing the test cases.

JUnit 5 provides us with two ***Timeout Assertions*** (see [here](https://howtodoinjava.com/junit5/junit-5-assertions-examples/#assertTimeout-assertTimeoutPreemptively)), which are used to test long running tasks. If given task inside testcase takes more than specified duration, then test will fail.

**Subtask 01**: The method, *squareRoot*, is expected to be returned within 1 second. However, due to having an endless loop, *squareRoot* will never finish. Please choose a right assertion to make a Timeout Testing for *squareRoot*. Give your answer in the testSquareRoot method in task02.CalculatorTest.

***Exception Test***. Some test cases are used to test the execution of the supplied Executable throws an exception of the *expectedType* and returns the exception.

**Subtask 02**: Dividing by zero will lead to an *ArithmeticException*. It means the *divide* method in task02.Calculator has a bug if it does not throw an *ArithmeticException* in this scenario. Please write a test method to check if the *divide* method can throw an *ArithmeticException* when the *divisor* parameter is zero. Give your answer in the testDivide method in task02.CalculatorTest.

### Task 03: Parameterized Test

[This video](https://youtu.be/srJ91NRpT_w) introduced the roman numeral problem. We provide its implementation in *task03.RomanNumeral* and its corresponding test class in *task03.RomanNumeralTest*. The method, *singleDigit*, tries to check the 7 roman numerals, I, V, X, L, C, D, and M, can be correctly mapped by *singleDigit* to their corresponding Arabic numbers, 1, 5, 10, 50, 100, 500, and 1000. However, the testing codes seem very duplicated and repetitive. Thus, it’s very difficult to perform large-scale test cases in this way. Fortunately, we can greatly simplify the testing codes by using ***Parameterized Test*** provided by JUnit 5. Please refer to user guide [here](https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests) to rewrite the test method *singleDigit* by using *Parameterized Test* in *task03.RomanNumeralParamTest*.

# Lab03: Testing Principles

## Exercise 01: failure, fault, or error

Having a certain terminology helps testers to explain the problems they have with a program or in their software. Below is a small conversation. Fill each of the caps with: failure, fault, or error.

**Mark**: Hey, Jane, I just observed a (1) \_ \_ \_ \_ \_ \_ in our software: if the user has multiple surnames, our software doesn’t allow them to sign in.

**Jane**: Oh, that’s awful. Let me debug the code so that I can find the (2) \_ \_ \_ \_ \_ \_.

*(a few minutes later)*

**Jane**: Mark, I found it! It was my (3) \_ \_ \_ \_ \_ \_. I programmed that part, but never thought of this case.

**Mark**: No worries, Jane! Thanks for fixing it!

Your Answer is ?

## Exercise 02: testing principle

Kelly, a very experienced software tester, visits Books!, a social network focused on matching people based on books they read. Users do not report bugs so often; Books! developers have strong testing practices in place. However, users do say that the software is not really delivering what it promises. What testing principle applies to this problem?

Your Answer is ?

## Exercise 03: testing principle

Suzanne, a junior software testing, just joined a very large online payment company. As a first task, Suzanne analyzed their past two years of bug reports. Suzanne observes that more than 50% of bugs have been happening in the ‘International payments’ module.

Suzanne then promises her manager that she will design test cases that will completely cover the ‘International payments’ module, and thus, find all the bugs.

Which of the following testing principles might explain why this is not possible?

1. Pesticide paradox.
2. Exhaustive testing.
3. Test early.
4. Defect clustering.

Your Answer is ?

## Exercise 04： only unit testing?

John strongly believes in unit testing. In fact, this is the only type of testing he actually does at any project he’s in. All the testing principles below, but one, might help in convincing John that he should also focus on different types of testing.

Which of the following is the least related related to help John in moving away from his ‘only unit testing’ approach?

1. Pesticide paradox.
2. Tests are context-dependent.
3. Absence-of-errors fallacy.
4. Test early.

Your Answer is ?