Lab Report for SQAT

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

**Before submitting your labwork, please DO read the submission instructions carefully** [**How-to-submit-your-labwork.docx**](How-to-submit-your-labwork.docx) **!**

# Lab01: 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 ?

# Lab02: JUnit for Unit Test

## 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/) **4/5**:
* Assert Functions, e.g., assertTrue, assertFalse.
* JUnit 4/5 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***!).
* To know about **Maven** and Maven-based Project
* To understand how to **add dependencies** by Maven
* To understand **Timeout Testing** and **Exception Test**
* To understand **Parameterized 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: Terminology Illustration

Please READ Lectures/Lec03-JUnit/junit.pdf 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 Statement**. There are two forms of the assert statement. Please illustrate what they are.

### Task 02: Config JUnit 4/5 in a Maven Project

Please read *Lectures/Lec03-JUnit/JUnit-Maven.docx* carefully. According to the instructions and steps illustrated in this .docx file, do the following subtasks:

Configure your own maven project by adding JUnit 4 and JUnit 5 dependencies;

Copy all of the Java class and test code to this project;

Run all of the test methods and snapshot the testing results.

NOTE: You should submit your maven project and your running results.

Now, please illustrate why we manage JUnit 4 and JUnit 5 dependencies by maven, rather than manually?

**Assert** is very important for Making Tests. Please read and run the test codes, CalTest in *Lectures/Lec03-JUnit/code/HelloMaven/src/test/java/lec03/junit/junit4/****CalTest.java***, and illustrate why? That is, if we don’t use Assert functions, what will happen in our testing code?

### Task 03: Parameterized Test.

[This video](https://youtu.be/srJ91NRpT_w) introduced the roman numeral problem. We provide its implementation in ***RomanNumeral.java*** and its corresponding test class in ***RomanNumeralTest.java***. The two Java classes are in: */Labwork/Lab02/hellojunit/src/main/java/roman/*. The method *singleDigit* in *RomanNumeralTest.java* 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*.

### Task 04: Test your own code with JUnit 5.0

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 |

# Lab03: Specification-Based Testing

Note: all of the following 7 exercises can be found [here](https://mordeky.github.io/SQAT/chapters/testing-techniques/specification-based-testing.html#exercises). You just need to report your answer in the corresponding subsection.

## Exercise 01

## Exercise 02

## Exercise 03

## Exercise 04

## Exercise 05

## Exercise 06

## Exercise 07

# Lab04: Boundary Testing

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/testing-techniques/boundary-testing.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are 01, 03, 05, and 07.

## Exercise 01

## Exercise 03

## Exercise 05

## Exercise 07

# Lab06: Structural-Based Testing

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/testing-techniques/structural-testing.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are 01, 03, 05, and 07.

## Exercise 01

## Exercise 03

## Exercise 05

## Exercise 07

# Lab07: Model-Based Testing

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/testing-techniques/model-based-testing.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are 08, and 09.

## Exercise 08

## Exercise 09

# Lab08: Design by Contracts

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/testing-techniques/design-by-contracts.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are 01, and 03.

## Exercise 01

## Exercise 03

# Lab09: Testing Pyramid

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/pragmatic-testing/testing-pyramid.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are 01, and 02.

## Exercise 01

## Exercise 02

# Lab10: Mock Objects

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/pragmatic-testing/mock-objects.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are 02, and 04.

## Exercise 02

## Exercise 04

# Lab11: Design for Testability

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/pragmatic-testing/design-for-testability.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise number is only 01.

## Exercise 01

# Lab12: Test-Driven Development

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/pragmatic-testing/tdd.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are only 01 and 04.

## Exercise 01

## Exercise 04

# Lab13: Test Code Quality and Engineering

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/pragmatic-testing/test-code-quality.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise numbers are only 01 and 05.

## Exercise 01

## Exercise 05

# Lab14: Mutation Testing

Note: all of the following exercises can be found [here](https://mordeky.github.io/SQAT/chapters/intelligent-testing/mutation-testing.html#exercises). You just need to report your answer in the corresponding subsection. Please notice the exercise number is only 01.

## Exercise 01