SENG 275 ASSIGNMENT 3 (6%)

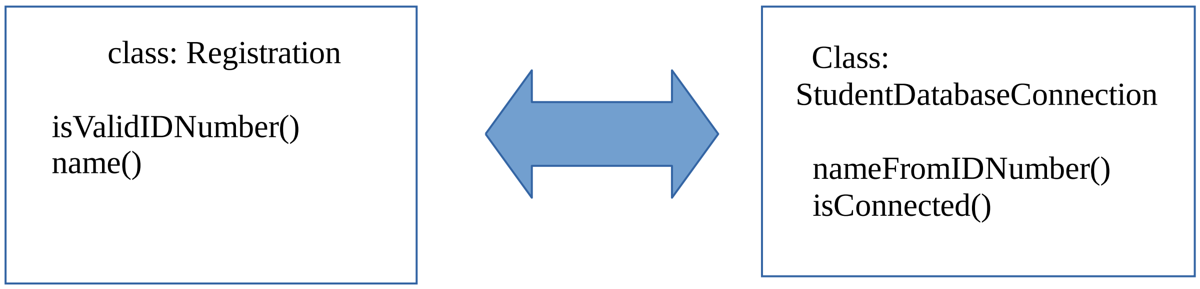
Instructor: Dr. Navneet Kaur Popli
Date: 19 July 2023
Submit by: 28 July 2023
Total Marks: 44 M

This is an individual activity. Copying, discussions with peers or cheating of any kind is not allowed. Plagiarised content is prohibited. Supplement with code screenshots, diagrams, tables, graphs, and charts wherever applicable.

Q1) You have been provided with an Assignment 3 code/repository named a3 on brightspace under assignments, which contains the application and test files necessary for this question. Please push assignment 3 on GitLab when you complete it. (30M)

Student numbers at Uvic are formatted according to a series of rules. Each student number begins with an uppercase V and is followed by exactly eight digits.

It is sometimes necessary to look up the name of a student, given only a student number. To do so, the software system below can be used, which has been partially implemented in the file **lib/src/main/java/a3/a3.java**. The outlines of tests have been created for you in **lib/src/test/java/a3/A3Test.java.**



Users of the Registration class can provide a Uvic ID number (as a string) to name(), and will get a string back containing the student's First Name, followed by a space, followed by their last name.

You have been asked to test the two functions in the Registration class. Unfortunately, the code is in an unfinished state. Specifically, the StudentDatabaseConnect class exists only as an interface, and the database itself won't be ready for some time. Additionally, the name() function is incomplete – it does not yet fulfill its Design by Contract obligations. Finally, there are no tests written yet.

You may assume that the following are valid students, and would be in the database if it were connected:

Student Number	Student name
V11111111	Andy Apple
V22222222	Betty Banana
V33333333	Cathy Cantaloupe
V44444444	Donald Durian

1. You must identify the code under test and set up one or more Test Doubles correctly. Use the Mockito framework for this and refer to Lab 6 as necessary. (8 marks)

2. You must complete the Registration.name() function, implementing its pre- and post- conditions correctly using Design by Contract principles. Refer to Lab 7 part 1 as necessary. You may not modify any other code in **a3.java**; you may only add the code to Registration.name() that is necessary to fulfill the contract. (4 marks)

3. You must write non-property (regular individual or parameterized Junit) tests to test the following behavior:

- Does name() have the expected behavior when the database is not connected? (2 marks)
- Does name() have the expected behavior when the database is connected, but the IDs do not fit the format of Uvic user IDs? Think about the rules above, and what sorts of edge cases there are. (3 marks)
- Does name() function correctly for students who are in the database? (In other words, are in the table above?) (3 marks)
- Does name() function correctly for valid student numbers that have no corresponding student in the table above? (In other words, the student numbers are formatted correctly, but do not appear in the table). (2 marks)

You will need to read over the specifications for the functions in **a3.java** to determine the correct expected behavior for these tests. Each of these tests must establish not only that you obtained the correct output from the function, but also that the system performed in the correct way – in other words, you must use Mocks rather than Stubs (and therefore you must verify() that the function calls you expected occurred, and that *no other interactions with your test doubles occurred*).

4. You must write property tests (using Jquik) to test the function Registration.isValidIDNumber(). Note that this is a static function, so you won't need test doubles or verify calls to complete this task. Specifically, you must:

- Write a test that checks one thousand valid Uvic ID numbers, generated randomly, and confirms that the function accepts them. (4 marks)
- Write a test that checks one thousand invalid Uvic ID numbers, each of which is made up of a **lowercase** letter followed by a number from 10000000 to 99999999. The sections of the Jquik user guide at https://jqwik.net/docs/current/user-guide.html on parameter generation will be helpful here. (4 marks)

You will complete your work for part 2 above in **a3.java**, and for parts 1, 3 and 4 in **A3Test.java**, in the sections indicated by comments.

You must submit as part of your assignment 3 pdf:

- A screenshot of your finished Registration.name() function from **a3.java**, showing the additions you made to fulfill the pre- and post-conditions.
- A screenshot of your code from **A3Test.java**, showing your tests. Take multiple screenshots if necessary to show all your tests.
- A screenshot of your **test results** from IntelliJ (in the bottom left corner of your screen), showing all your tests passing. Please expand tests that have been grouped together using the Expand All button above the tests results window.

Note: Please contact shujamughal@uvic.ca for any clarification regarding the above question.

Q2) Consider a function called ComplexAddition() which is part of the Calculator class used in the lectures on mutation testing. Submit the answer in Brightspace as a pdf file. (14M)

```
public int ComplexAdd(int a, int b)
{
        if (a<2)
                        {
                                return (a+b) * -1;
                        }
else
                        {
                                return a+b;
                        }
}
```

a) Write JUnit test cases to give 100% line coverage. (4M)
b) Perform Mutation testing on this function. List all mutations done by the system. Submit the screenshot of Mutations. (2M)
c) Write additional JUnit test cases to strengthen your test suite to kill all mutants. Specify which test case kills which mutant. Submit screenshots after killing each mutant. (6M)
d) What is the final test suite? Are you able to achieve 100% mutation coverage? (2M)