SENG 275 ASSIGNMENT 1 Summer 2023 (7%)-O5

Instructor: Dr. Navneet Kaur Popli
Assigned Date: 1 June 2023
Submit by: 10 June 2023
Total Marks: 56

This is an individual activity. Copying, discussions with peers or cheating of any kind is not allowed. Plagiarised content is prohibited. Grades will be deducted for plagiarism/cheating.

Q1) Following are the specifications for a Roman numerals Program:

"Implement a program that receives a string as a parameter containing a roman number and then converts it to an integer."

In roman numerals, letters represent values:

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

Letters can be combined to form numbers. For example we make 6 by using 5 + 1 = 6 and have the roman number VI Example: 7 is VII, 11 is XI and 101 is CI. Some numbers need to make use of a subtractive notation to be represented. For example we make 40 not by XXXX, but instead we use 50 - 10 = 40 and have the roman number XL. Other examples: 9 is IX, 40 is XL, 14 is XIV.

The letters should be ordered from the highest to the lowest value. The values of each individual letter is added together. Unless the subtractive notation is used in which a letter with a lower value is placed in front of a letter with a higher value.

Combining both these principles we could give our method MDCCCXLII and it should return 1842.

A **possible** implementation for the Roman Numerals requirement is as follows:

```
public class RomanNumeral {
 private static Map<Character, Integer> map;
 static {
  map = new HashMap<>();
  map.put('I', 1);
  map.put('V', 5);
  map.put('X', 10);
  map.put('L', 50);
  map.put('C', 100);
  map.put('D', 500);
  map.put('M', 1000);
 }

 public int convert(String s) {
  int convertedNumber = 0;

  for (int i = 0; i < s.length(); i++) {
   int currentNumber = map.get(s.charAt(i));
   int next = i + 1 < s.length() ? map.get(s.charAt(i + 1)) : 0;

   if (currentNumber >= next) {
    convertedNumber += currentNumber;
   } else {
    convertedNumber -= currentNumber;
   }
  }

  return convertedNumber;
 }
}
```

a) With the implementation in hand (please refer to the note below to improve the implementation), the next step is to devise test cases for the program.

Use your experience as a developer to devise at least 10 different test cases which test the program efficiently.

Give reasons why these are testing the input domain completely (They should be able to form equivalent partitions of the input domain).
Provide meaningful names and descriptions to these tests.

Submit implementation code both in the assignment pdf file as text/intellij snapshot (Give a breif explanation of each snapshot) and as a zip file which can be executed.
(5+10=15M : 5M for improved implementation and 10M for test cases)

b) Create automated Junit tests for these test cases. Give at least 2 different inputs per test case, 1 which passes the test and 1 which fails the test. Confirm your results by pasting snapshots of implementations and the output of tests passing/failing. (10M)

Note: The implementation in the assignment writeup is just a "possible implementation" - as you note, it fails to detect some invalid inputs like 'XXXX', 'ABC', 'VV', 'DD', 'DCM'; lowercase inputs, empty inputs etc. You must write your own implementation by improving on the implementation already given to you. Your implementation should detect these invalid inputs, and handle them appropriately. The assignment doesn't specify how you should indicate that an invalid input has been received, so it is your job as a designer to figure out the best way of doing so. Probably an exception makes sense, but that is for you to decide.

Q2) Domain testing is a very important strategy for testing an input domain for an application.

Say you have to develop a game called 'CatchMeIfYouCan' which uses a 6X6 GameBoard for the players which run around the board catching each other and gaining points.

You have to develop a class called 'GameBoard' which creates a board and tests for some points to be in or outside the board (Which is basically to test whether a player is inside the board or outside).

The permitted x and y coordinate values for the board are 0,1,2,3,4,5 because 6X6 is the maximum sixe of the board. Perform the following activities: (Total: 25M)

a) Create a class called 'GameBoard' that sets up a 6X6 squre board in the form of a matrix for the game before the entire test suite. Use the correct annotations. (5M)

b) Start by generating 5 good weather test cases and provide specific test inputs to those. These input points in the board are in the form of (x,y) coordinates to assert that the players are within the board. Also generate 4 bad weather input points in the board in the form of (x,y) coordinates to test that the player is outside the board.

Now test the boundaries. Generate at least 8 test cases, 2 for each boundary. One on and one off point for each boundary (You may wish to test the intersections but they will not be graded in this question). Explain the reason for your choice. Generate error message/ exception handling when player goes outside the board. (5+4+8+1+2=20M)

Submit both in pdf file (As snapshots in intellij. Give a breif explanation of each snapshot) and executable code in a zip file in Brightspace.

Q3) Consider a program, which takes as input, three integers, x, y, and z, each representing the length of one side of a triangle, and classifies the triangle as one of the following:

- *invalid*: at least one side does not have a positive length, or one side is greater than the sum of the other two sides.
- *equilateral*: all sides are the same length.
- *isosceles*: two sides are the same length; or
- *scalene*: all sides are of different length.

A possible implementation of this program is shown below: (You can improve upon it if required)

```
type enum Triangle = {equilateral, isosceles, scalene, invalid}

Triangle categorise(int x, int y, int x)

public class Triangle
{

public static String TriangleType(int x, int y, int z)
{
    if (x > 0 && y > 0 && z > 0 &&
        x + y >= z && x + z >= y && y + z >= x)
       {
        if (x == y)
            if (y == z) return "equilateral";
            else return "isosceles";
        else if (y == z) return "isosceles";
        else if (x == z) return "isosceles";
        else return "scalene";
       }
    else return "invalid";
}
}
```

(a) Identify Boundary conditions for **equilateral triangle** condition testing. Test boundaries between equilateral triangle and scalene, equilateral triangle and isoceles and equilateral triangle and invalid triangle by 1 on and one off point for each boundary. Please give specific data values in the form of (x,y,z) along with explanation of the choice of these values.(3M)

(b) Create automated Junit tests for these test cases. Confirm your results by pasting snapshots of the output of test passing/failing in addition to executables as a zip file.(3M)