INSTRUCTIONS :

1. Complete all questions in your designated project group.
2. All members must contribute to writing the codes. (i.e. 1 question = 1 person, and share the workload if there's an additional question relative to the actual number of members in your team (i.e. 5)). Ensure that all members must understand and explain codes from any of the questions.
3. During viva, all students in each team will be randomly asked to describe, answer and edit any of the answers provided. Marks will be given to your ability to present the answers.

Lab Report

Prepare a report to solve the above problems. The report should contain all the sections as below for each question:

Section	Description
1. Problem	Description on the problem
2. Solution	Explanation on how to solve the problems below
3. Sample Input & Output	A few sets of input and output (snapshot)
4. Source code	Java source code

Requirements

1. Group Assignment (Grouping is the same as your project group)
2. Cover page that includes all student matric number and full name.
3. Font: Times New Roman 12, Line Spacing: 1 ½ Spacing
4. Submit to Spectrum according to your OCC. **Deadline** : Before your viva session (W7).

Question 1: Longest Palindrome

Problem Statement:

Write a method called ***findLongestPalindromicSubstring*** to find the longest palindromic substring in a given string. A *palindromic substring* is a substring of the given string that reads the same forwards and backwards. The method should return the longest palindromic substring found within the input string.

In the ***main*** method, get input from the user and then display the longest palindromic substring within the input string using the ***findLongestPalindromicSubstring***

The following is given:

- The input string contains all lowercase letters
- The input string does not contain any special characters or whitespaces

- If there exists more than one unique longest palindromic string, your program may return any of them

Hint:

- Make sure to check for both odd length and even length palindromes

Sample Input Output:

Example 1
Enter string: ***bananas*** *Longest Palindromic Substring: anana*

Example 2
Enter string: ***racecar*** *Longest Palindromic Substring: racecar*

Example 3
Enter string: ***abaabba*** *Longest Palindromic Substring: abba*

Question 2 : Merge into Array

Problem Statement

Construct a Java program that receives two lines of strings from the user and interprets them as integer arrays. You should use a method called parseArray, which takes the input and returns the array. Then, construct another method called mergeArray, which combines the two arrays into one array, while sorting all integers in ascending order and removing any repeated values. Print the final merged array. You may construct any additional methods that may help you in solving the problem.

Sample Input	Sample Output
Array 1: 1, 2, 3 Array 2: 6, 3, 4	[1, 2, 3, 4, 6]
Array 1: 5, 2, 1, 3, 8 Array 2: 8, 1	[1, 2, 3, 5, 8]

Note: You are not allowed to use the ArrayList.

Question 3: Romans

Problem Statement:

In the mid-lecture break of Computing Mathematics II, Kah Sing chatted about something that he found very interesting to his friends Ridwan and Suresh in the lecture hall.

Kah Sing: '*Eh*, *Wan*, *Su*, ever wondered why Dr Soh doesn't name this Computing Mathematics module IIIV, IIIVX, IIIVXXXXL instead? There are so many ways to write it *sia*!'

Ridwan: 'Haha, of course not, *kan*! Imagine memorising Roman characters this long just for the number "2"!'

Suresh: '*Deyh*, you're right, but actually *hor*, *Sing*'s also got a point, *tau*! Our KK8 room G101 always has *Sing* with brilliant ideas one *lah*!'

Kah Sing: '*Hmm*, why not I write a program to test our understanding of Roman numerals after class? Who knows, maybe it'd help us in our study for this module?'

The Roman numerals, originating from ancient Rome and widely used until the Late Middle Ages in Europe, employ letters from the Latin alphabet to represent numbers. Each letter has a fixed value, with the modern system using only seven letters: I, V, X, L, C, D, and M; each of which represents each of the integers 1, 5, 10, 50, 100, 500, 1000.

Many people misunderstand Roman numerals, thinking they always follow a subtractive syntax. In this system, a smaller digit before a larger one is subtracted from the larger one. However, Romans may opt for alternate systems instead (like what Kah Sing suggested)! Romans themselves only really used the subtractive syntax for smaller numbers, as it was largely avoided for larger numbers to give more clarity (or even for smaller numbers purely for simplicity, like what Ridwan and Suresh brought up).

Requirements

1. Under the class ***Romans***,
   a. Create a method named ***generateInitials*** to generate the initials of the user(s) of the computer program.

   b. Create a method named ***convertArabics*** to convert the given roman numeral (which only contains I, V, X, L, C, D, M) to Arabic form (which only contains 1, 2, 3, 4, 5, 6, 7, 8, 9, 0).

   c. Create a method named ***convertRomans*** to convert the given Arabic numeral to roman form.

2. Ensure there is no error in compiling, running and handling invalid inputs in the code, and when sample inputs and outputs are passed in, there is no error found by the grader.

3. Ensure your report is done and everything you include in the report aligns with all the aspects of your code.

For all notes on input and output, refer to the following pages.

Input
The first line is the name(s) of the program user(s). The second line is an integer ***n***, between 1 inclusive and 10 inclusive. Then, for the next ***n*** lines, the Roman numeral statements are typed in, each shall be in the format of '***roman_numeral_1 arithmetic_operand roman_numeral_2 equality_sign roman_numeral_3***', such as 'I + II = III' and 'I & II ? III', the former of which is valid while the latter is invalid. The seven letters to represent integers can either be in uppercase or lowercase. The arithmetic operands only include addition ('+'), subtraction ('-'), multiplication ('*'), division ('/'), modulo ('%'), and exponentiation ('^'). For division, in the case of a non-integer answer, the roman_numeral_3 shall return an integer value, which is the same as the quotient of the operation.

Note for this problem, any digit written to the left of a larger digit in the input will be subtracted, even if they are not directly adjacent.

Note, also, that for a statement to be valid, each of the ***roman_numeral_1, roman_numeral_2, and roman_numeral_3*** has to be within the range of 1 inclusive and 3999 inclusive.

For example, even if the statement seems correct, such as 'D % C = O', which in the Arabic numeral form is '500 % 100 = 0', with the substituting of the 'O' character for the number 0 since there is no such representation in the Roman numeral system, but as the ***roman_numeral_3***, which in this case is the 'O' character, is invalid, the statement shall be deemed as invalid.

Note, as well, that if a Roman numeral statement input is invalid, it counts as a wrong statement.

Output
The first line is in the format of 'Statements for the Roman numeral test are sent in by' followed by the initials of the name(s) (**Note:** only 'bin', 'binti', 'a/l', a/p', 'al', 'ap', '@' are not represented in the initials as parts separated by delimiters), then '**, (**', which is followed by the name typed by the program user, appended by '**).**'. For the name of the user, the delimiters are the ' ' (whitespace), '**-**' (dash), '**'**' (apostrophe), '**_**' (underscore), '**.**' (full stop), and '**,**' (comma) characters only. After that, print one empty line.

For the next ***n*** lines, for a valid Roman numeral statement input, the output shall be in the format of '***arabic_numeral_1 arithmetic_operand arabic_numeral_2 equality_sign arabic_numeral_3***', if the nth statement input by the user is correct append by 'is correct.' on the same line; else append by ' is wrong, as the answer should be

roman_numeral_answer, which is ***arabic_numeral_answer*** in Arabic numeral form.' on the same line. If the Roman numeral statement is invalid, print 'Invalid Statement.' instead.

After that, print one empty line, then print on the next line 'Number of Correct Statements = ' followed by the number of correct statements input by the user. The next line shall print 'Percentage of Correct Statements = ' followed by the percentage of correct statements input by the user, appended by an '%' (ampersand) character on the same line.

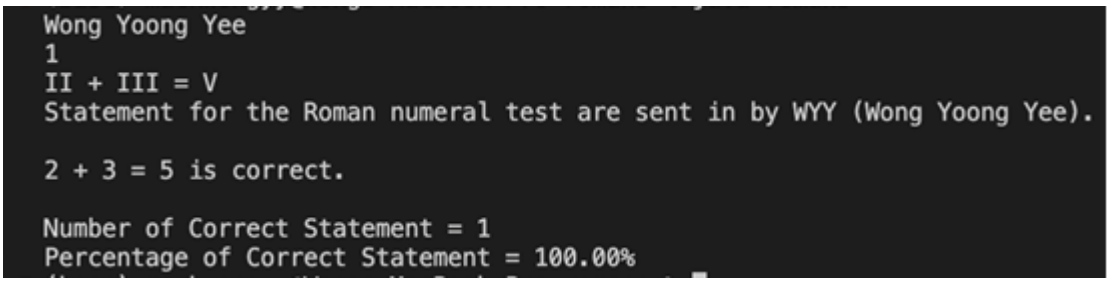
Note, that the '***,***' (comma) character can only be a delimiter if and only if it is, by itself, a separate part in the line of name(s). For example, the name 'Lee, Kah Sing' shall print the initials 'LKS', whereas the names 'Ridwan Faiz bin Mohamad Hassan , Suresh a/l Subramaniam' shall print the initials 'RFMH , SS'.

Note, also, the ***roman_numeral_answer*** is in the modern Roman numeral system, such as for the number 99, it should be XCIX instead of IC in the subtractive Roman numeral system or those in some other Roman numeral system.

Note, as well, that the validity of a statement has a higher precedence than the correctness of the statement, thus the printing of 'Invalid Statement.' has a higher precedence than '***arabic_numeral_1 arithmetic_operand arabic_numeral_2 equality_sign arabic_numeral_3*** is correct.' and '***arabic_numeral_1 arithmetic_operand arabic_numeral_2 equality_sign arabic_numeral_3*** is wrong, as the answer should be ***roman_numeral_answer***, which is ***arabic_numeral_answer*** in Arabic numeral form.'.

For example, given input 'M ^ II = M', the output shall be 'Invalid Statement.', as the correct answer in Arabic numeral form should be 1000000, which has exceeded the set limit of 3999, as the statement is invalid first before it is wrong; likewise for statements which are correct but contain values which lie outside the valid range of numeral between 0 and 3999.

Sample Input/Output



```
(base) mackwongyy@Wongs-MacBook-Pro romans % javac romans.java
(base) mackwongyy@Wongs-MacBook-Pro romans % java romans
Lee Kah Sing , Ridwan Faiz bin Mohamad Hassan , Suresh a/l Subramaniam
10
XXIV + MMCD = MMCDXXIV
XXIV - MMCD = XXIVMMCD
MMCD - XXIV = MMCCCLXXVI
XX * iv = lxXX
MM / CD = V
CCcc % CcCcCcCcCcCcCcCcCcCc = CCCC
V ^ V = MMMXXV
MDCLXVI ^ I = MDCLXVI
MDCLXVI - IVXLCDM = MCCCXXXII
MDCLXVI = IVXLCDM + MCCCXXXII
Statement for the Roman numeral test are sent in by LKS, RFMH, SS (Lee Kah Sing , Ridwan Faiz bin Mohamad Hassan , Suresh a/l Subramaniam).

24 + 2400 = 2424 is correct.
Invalid Statement.
2400 - 24 = 2376 is correct.
20 * 4 = 80 is correct.
2000 / 400 = 5 is correct.
400 % 2000 = 400 is correct.
5 ^ 5 = 3025 is wrong, as the answer should be MMMCXXV, which is 3125 in Arabic numeral form.
1666 ^ 1 = 1666 is correct.
1666 - 334 = 1332 is correct.
Invalid Statement.

Number of Correct Statements = 7
Percentage of Correct Statements = 70.00%
```

Question 4: Number Formatter

Problem statement:

Write a function called *format* that takes an integer *number* and an integer *width* as input. The function should return a string representation of the number, padded with zeros on the left to reach the specified width. If the number is already longer than the width, the function should return the original string representation of the number.

Here's an example of how the function should work:

- *format(34, 4)* should return *"0034"*.
- *format(34, 5)* should return *"00034"*.
- *format(34, 1)* should return *"34"*.

Then, write a program that prompts the user to enter an integer and a width, then calls the format function and displays the resulting string.

Question 5: Fruit Store Billing System

Problem Statement:

Write a Java program that implements a billing system for a fruit store, FreshMart. The program will display the available fruits, take user input to select a fruit and quantity, check the stock availability, and generate a bill.

You are provided the **`main method`**, which handles user input and basic program flow. Your job is to complete the remaining methods so that the program behaves as expected.

```
import java.util.Scanner;

public class freshMart {
```

```
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        String[] fruits = {"Apple", "Grape", "Banana", "Mango", "Orange", "Strawberry",
"Blueberry"};
        int[] storage = {45, 57, 75, 34, 75, 65, 78};
        double[] price = {5.00, 10.00, 2.50, 6.00, 3.00, 15.00, 12.00};

        printFruit(fruits);

        System.out.print("Please enter the fruit you want to buy: ");
        String fruitSelect = in.nextLine();

        System.out.print("Unit or Box: ");
        int quantity = in.nextInt();

        int fruitIdx = getIdxFruit(fruits, fruitSelect);

        if (!haveProduct(fruits, fruitIdx)) {
            System.out.println("Sorry we currently do not offer this product.");
        } else if (!haveStock(fruitIdx, storage, quantity)) {
            System.out.println("Sorry, we do not have enough stock for " + fruits[fruitIdx]
+ ". Please come again later!");
        } else {
            System.out.println("\nKindly Checkout here: ");
            generateBill(fruits, fruitIdx, price, quantity);
        }
    }
}
```

Your Task:

You are required to **implement the following methods** to complete the program:

a. **`printFruit(String[] fruits)`:** This method should display the list of available fruits in the store.
b. **`getIdxFruit(String[] fruits, String selectedFruit)`:** This method should return the index of the selected fruit in the array. If the fruit is not found, it should return -1.
c. **`haveProduct(String[] fruits, int fruitIdx)`:** This method checks if the selected fruit is available in the store.
d. **`haveStock(int fruitIdx, int[] storage, int quantity)`:** This method checks if the selected fruit has enough stock to meet the customer's requested quantity.
e. **`generateBill(String[] fruits, int fruitIdx, double[] price, int quantity)`:** This method generates and prints the bill, showing the product, quantity, price per unit, and total price.

Expected Output:

```
FreshMart Fruit Selection
1. Apple
2. Grape
3. Banana
4. Mango
5. Orange
6. Strawberry
7. Blueberry
Please enter the fruit you want to buy: orange
Unit or Box: 23
```

```
Kindly Checkout here:
------------------------------------------------------------
-----------------Fresh Mart Receipt-------------------
------------------------------------------------------------
Product                 Quantity  Price per Unit(RM)
Orange                       23                3.00
------------------------------------------------------------
Total Price: RM69.00
------------------------------------------------------------
```

Question 6: Matrix Manipulation

Problem Statement:

In this problem, you will write code that allows you to do the following:

i. Take input of ***n x n*** integers to form a matrix (2D array)
ii. Rotate the matrix by 90 degrees clockwise
iii. Display the original matrix and the matrix after rotating it

Implementation:

1. **Method 1: *parseMatrix***
   a. Write a method called ***parseMatrix*** that prompts the user to enter an integer **n.** This will be the dimensions of the ***n x n*** matrix.
   b. Next, prompt the user to enter the ***n*** rows of the matrix each with ***n*** integers to form the matrix.
   c. Store the input matrix as a 2D array for further processing
   d. It is given that:
      i. The user will always input an integer
      ii. The user will only enter exactly ***n x n*** integers in total
      iii. The user will input number starting from first row, from left to right
      iv. ***n*** is a number between 0 and 100

2. **Method 2: *displayMatrix***
   a. Write a method called ***displayMatrix*** that takes a 2D array as input and outputs the array in the form of a matrix (refer to the sample input and output for examples).
   b. Use this method to display the input matrix we obtained from the ***parseMatrix*** method and to ensure that your ***parseMatrix*** method correctly retrieves the matrix.

3. **Method 3: *rotateMatrix***
   a. Write a Java method ***rotateMatrix*** that takes a 2D array (matrix) as input and rotates the matrix 90 degrees clockwise. The method should rotate the matrix in place without returning anything.
   b. Use this method to rotate the input matrix and then use the ***displayMatrix*** method to display the matrix after it has been rotated.

Sample input output:

Example 1

```
Enter the dimension of the square matrix: 3
Enter row number 1: 1 2 3
Enter row number 2: 4 5 6
Enter row number 3: 7 8 9

The original matrix:
1 2 3
4 5 6
7 8 9

The matrix after rotating:
7 4 1
8 5 2
9 6 3
```

Example 2

```
Enter the dimension of the square matrix: 4
Enter row number 1: 1 2 3 4
Enter row number 2: 5 6 7 8
Enter row number 3: 9 10 11 12
Enter row number 4: 13 14 15 16

The original matrix:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

The matrix after rotating:
13  9  5  1
14 10  6  2
15 11  7  3
16 12  8  4
```