Viva 2 Lab Report

UNIVERSITI MALAYA

Name	Matric Number
Lee Zhen Xue	24004491
Too Yun Jie	24004472
Wang Kok Ming	24004533
Tay Jia Le	24067892
Wang Li Xin	24067867

Q1 Longest Palindrome

PROBLEM

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

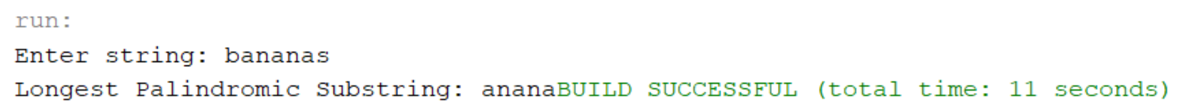
• The input string does not contain any special characters or whitespaces

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

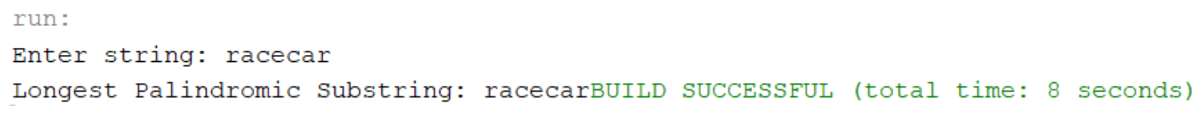
SOLUTION

1. Prompt user to enter a String.
2. Start from index 1, make a substring from index 1 to the index 2.
3. Reverse the substring and compare it to the original substring to check whether it is a palindromic.
4. If it is palindromic, compare the length of the substring and check whether it is the longest.
5. Repeat step 2 but the substring is from index 1 to index 2, 3, 4… until the last index.
6. Repeat step 5 but the substring is from index 2, 3, 4… until the last index.
7. Display the result.

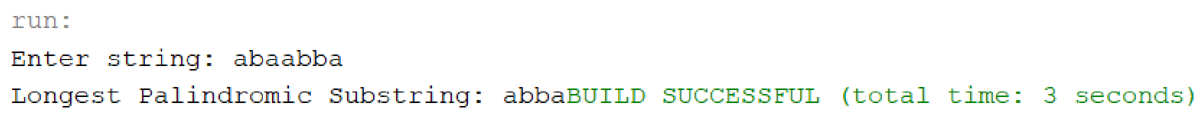
SAMPLE INPUT & OUTPUT



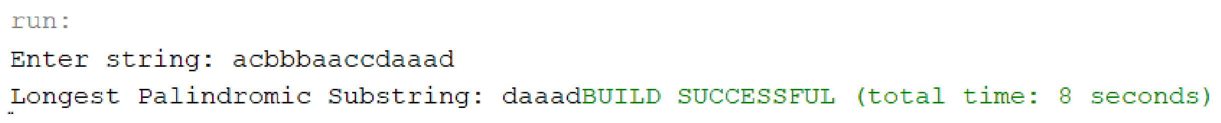
```
run:
Enter string: bananas
Longest Palindromic Substring: ananaBUILD SUCCESSFUL (total time: 11 seconds)
```



```
run:
Enter string: racecar
Longest Palindromic Substring: racecarBUILD SUCCESSFUL (total time: 8 seconds)
```



```
run:
Enter string: abaabba
Longest Palindromic Substring: abbaBUILD SUCCESSFUL (total time: 3 seconds)
```



```
run:
Enter string: acbbbaaccdaaad
Longest Palindromic Substring: daaadBUILD SUCCESSFUL (total time: 8 seconds)
```

SOURCE CODE

```
import java.util.Scanner;

public class Q1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter string: ");
        String input = sc.next();
        System.out.print("Longest Palindromic Substring: " + findLongestPalindromicSubstring(input));
    }

    public static String findLongestPalindromicSubstring(String word) {
        int wordCount = 0;
        String tempString;
        String reversedSubstring = "";
        String panlindromicSubstring = "";
        //Finding from the first index
        for(int startIndex = 0; startIndex < word.length(); startIndex++) {
            //Start from first index loop to the end to find whether there is panlindromic or not
            for(int endIndex = startIndex; endIndex <= word.length(); endIndex++) {
                tempString = word.substring(startIndex, endIndex);
            //reverse the string and compare it to the original string
            for(int index = tempString.length() - 1; index >= 0; index--) {
               reversedSubstring = reversedSubstring + tempString.charAt(index);
            }

                     if(reversedSubstring.equals(tempString)  &&  reversedSubstring.length() >=
wordCount){
               panlindromicSubstring = tempString;
               wordCount = reversedSubstring.length();
            }
            reversedSubstring = "";//reset the reversedSubstring for the next loop
         }
      }

      return panlindromicSubstring;
   }
}
```

Q2 Merge Into Array

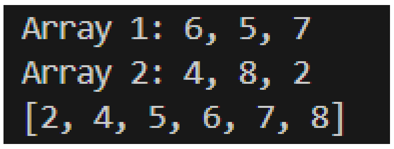
PROBLEM

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.

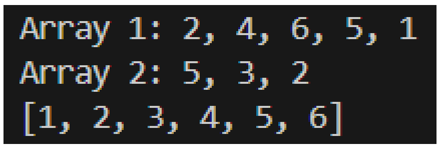
SOLUTION

1. Create a parseArray method to putting value inside an array of unknown size.
2. Create a mergeArray method to merge two array by making data distinct and sorted.
3. In main method, create two array and putting value into the array using parseArray method then merge it to mergedArray using mergeArray method.
4. Print mergedArray.

SAMPLE INPUT & OUTPUT



```
Array 1: 6, 5, 7
Array 2: 4, 8, 2
[2, 4, 5, 6, 7, 8]
```



```
Array 1: 2, 4, 6, 5, 1
Array 2: 5, 3, 2
[1, 2, 3, 4, 5, 6]
```

SOURCE CODE

```
import java.util.Arrays;
import java.util.Scanner;

public class Q2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Array 1: ");
        int[] array1 = parseArray(input);
        System.out.print("Array 2: ");
        int[] array2 = parseArray(input);
        input.close();
        int [] merged = mergeArray(array1, array2);
        System.out.println(Arrays.toString(merged));
    }

    public static int[] parseArray(Scanner input) {
        String[] inputArray = input.nextLine().replace(" ", "").split(",");
        int[] result = new int[inputArray.length];
        for (int i = 0; i < inputArray.length; i++) {
            result[i] = Integer.parseInt(inputArray[i]);
        }
        return result;
    }

    public static int[] mergeArray(int[] array1, int[] array2) {
        int[] mergedArray = new int[array1.length + array2.length];
        System.arraycopy(array1, 0, mergedArray, 0, array1.length);
        System.arraycopy(array2, 0, mergedArray, array1.length, array2.length);
        mergedArray = Arrays.stream(mergedArray).distinct().sorted().toArray();
        return mergedArray;
    }
}
```

Q3 Romans

<u>PROBLEM</u>

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.

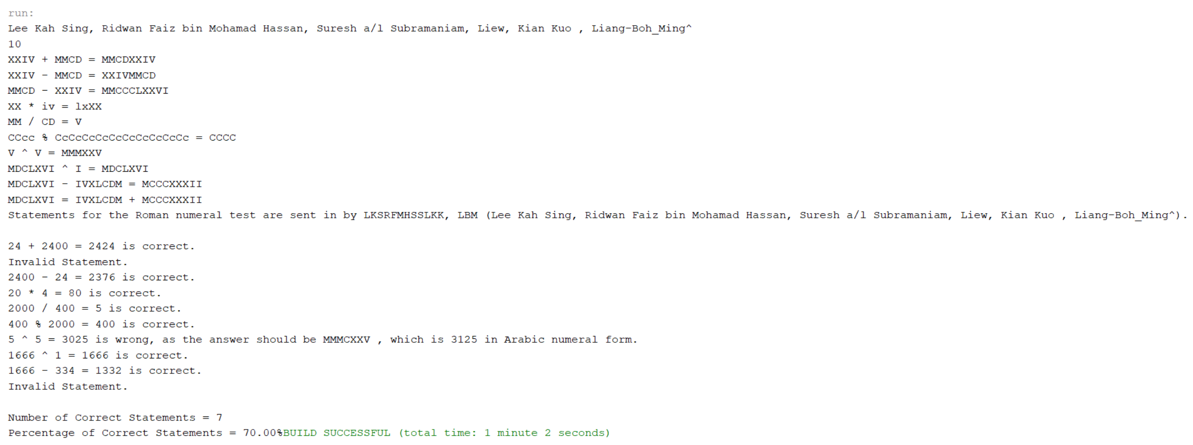
SOLUTION

1. Prompt user to input name(s) of user(s).
2. Split the input string using delimiters (,) and store them into an nameArray.
3. Prompt user to enter the number of equations, n
4. Prompt user to input n number of equations in roman numerals and store them as string in the array romanInput.
5. Call generateInitials method to generate initials for user(s).
    a. Take nameArray as an parameter and start the loop for each name in the nameArray.
    b. Split the name using delimiters specified (( ), (-), ('), (_), (.), (,)) and store them into an nameWords array.
    c. Start the loop for each nameWord inside the nameWords array
    d. Check whether it is a specific abbreviation that should be ignored ("bin", "binti", "a/l", "a/p", "al", "ap", "@").
    e. If it is not a specific abbreviation that should be ignored, Store the first letter of the nameWord as an uppercase letter inside the initials array.
    f. Repeat step (d) until all the nameWord is looped completely.
    g. Repeat step (a) for the second name in the nameArray and so on until the last name in the nameArray.
    h. Display the initials.
6. For each equation in the array romanInput, split the roman equation using the delimiter ( ) and store them in romanEquation.
7. Check whether the array of the equationEquation has a length of 5. If not, the equation is seen as invalid.
8. Loop through each element in the romanEquation and assign the value of roman_numeral_1, arithmetic_operand, roman_numeral_2, equality_sign, roman_numeral_3 respectively using the index starting from 0.
9. When assigning values for the variable, check whether they are valid or invalid.
10. Call the method convertArabics for each roman numeral.
    a. Take the roman numeral as a parameter and loop through every character in the roman numeral string.
    b. By using a switch case, determine whether the letter is 'I', 'V', 'X', 'L', 'C', 'D' or 'M' which represent 1, 5, 10, 50, 100, 500, 1000 respectively. Otherwise, return 0 which indicates it is an invalid roman numeral.

c. Using three variables to keep track of the last, current, and next roman numeral.
d. If the largest roman letter is in front of the roman numeral, treat it as normal roman numeral and use the normal roman numeral rules.
    i. A larger roman letter in the next of a smaller roman letter means the value of this roman numeral is the larger roman letter subtracted by the smaller roman letter.
    ii. Check for all the invalid forms of roman numerals.
e. If the largest roman letter is at the back of the roman numeral, read from behind and start with the value of the last roman letter, subtract it with all the roman letters in front of it.

11. After converting all the roman numerals to arabic. Calculate the outcome of the equation with roman_numeral_1, arithmetic_operand, roman_numeral_2 and check whether the answer given which is the roman_numeral_3 is correct or is incorrect.
12. If correct, display the equation as correct and increase the number of correct statements by 1.
13. If incorrect, display the correct answer for the equation and call the method convertRoman.
    a. Take the arabic integer as a parameter and initialise a variable called digits as 10. Using a loop, mod the arabic integer with 10 to get the least significant digit.
    b. Using the digit, determine what roman numeral should be used for it by using a switch case with digit and the least significant digit.
    c. Store it into the string called roman and add an empty space.
    d. Divide the arabic integer with 10 and multiply the digits by 10.
    e. After looping through every digit. Split the roman string by using empty space as delimiter and store each roman letter in the array.
    f. Reverse all the roman letters and combine them into a single string.
14. Display the number of correct statement and the percentage of correct statements.

SAMPLE INPUT & OUTPUT



```
run:
Lee Kah Sing, Ridwan Faiz bin Mohamad Hassan, Suresh a/l Subramaniam, Liew, Kian Kuo , Liang-Boh_Ming^
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
Statements for the Roman numeral test are sent in by LKSRFMHSSLKK, LBM (Lee Kah Sing, Ridwan Faiz bin Mohamad Hassan, Suresh a/l Subramaniam, Liew, Kian Kuo , Liang-Boh_Ming^).

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%BUILD SUCCESSFUL (total time: 1 minute 2 seconds)
```

SOURCE CODE

```
import java.util.Scanner;

public class Romans {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String nameString = sc.nextLine();
        String[] nameArray = nameString.split(" , ");

        int n;
        while(true){
            if(sc.hasNextInt()){
                n = sc.nextInt();
                break;
            }else {
                System.out.println("Please enter integer number only!");
                sc.nextLine(); //this is to consume the invalid input
            }
        }
        sc.nextLine();//clear the input

        //Roman Numeral Part
```

```java
String roman_numeral_1 = "";
String arithmetic_operand = "";
String roman_numeral_2 = "";
String equality_sign = "";
String roman_numeral_3 = "";

int arabic_numeral_1 = -1;
int arabic_numeral_2 = -1;
int arabic_numeral_3 = -1;

int numberOfCorrectStatement = 0;

String answerRoman;

String[] romanInput = new String[n];
String[] arabicOutput = new String[n];
boolean isInvalid = false;

for(int i = 0; i < romanInput.length; i++) {
   romanInput[i] = sc.nextLine();
}

System.out.print("Statements for the Roman numeral test are sent in by ");
String[] initials = generateInitials(nameArray);
for(int i = 0; i < initials.length; i++) {
   if(i == initials.length - 1)
      System.out.print(initials[i] + " ");
   else
      System.out.print(initials[i] + ", ");
}
System.out.print("(");
for(int i = 0; i < nameArray.length; i++) {
   if(i == initials.length - 1)
      System.out.print(nameArray[i] + ").\n");
```

```
    else
        System.out.print(nameArray[i] + " , ");
}
System.out.println();



for(int i = 0; i < romanInput.length; i ++) {
    isInvalid = false;
    String[] romanEquation = romanInput[i].split(" ");

    if(romanEquation.length != 5) {
        arabicOutput[i] = "Invalid statement.";
        continue; //invalid
    }

    for(int k = 0; k < 5; k++) {
        switch(k) {
            case 0:
                roman_numeral_1 = romanEquation[k];
                arabic_numeral_1 = convertArabics(roman_numeral_1);
                if(arabic_numeral_1 == 0) {
                    isInvalid = true;
                }
                break;
            case 1:
                arithmetic_operand = romanEquation[k];
                break;
            case 2:
                roman_numeral_2 = romanEquation[k];
                arabic_numeral_2 = convertArabics(roman_numeral_2);
                if(arabic_numeral_2 == 0) {
                    isInvalid = true;
                }
                break;
        case 3:
            equality_sign = romanEquation[k];
            if(!equality_sign.equals("=")){
                isInvalid = true;
            }
            break;
        case 4:
            roman_numeral_3 = romanEquation[k];
            arabic_numeral_3 = convertArabics(roman_numeral_3);
            if(arabic_numeral_3 == 0) {
                isInvalid = true;
            }
            break;
    }
}

if(isInvalid) {
    arabicOutput[i] = "Invalid Statement.";
    continue;
}
int answerArabic;

switch(arithmetic_operand) {
    case "+":
        answerArabic = arabic_numeral_1 + arabic_numeral_2;
        break;
    case "-":
        answerArabic = arabic_numeral_1 - arabic_numeral_2;
        break;
    case "*":
        answerArabic = arabic_numeral_1 * arabic_numeral_2;
        break;
    case "/":
        answerArabic = arabic_numeral_1 / arabic_numeral_2;
```

```
            break;
          case "%":
            answerArabic = arabic_numeral_1 % arabic_numeral_2;
            break;
          case "^":
            answerArabic = (int)Math.pow(arabic_numeral_1, arabic_numeral_2);
            break;
          default:
            arabicOutput[i] = "Invalid Statement.";
            continue;
        }

              arabicOutput[i] = (arabic_numeral_1 + " " + arithmetic_operand + " " +
arabic_numeral_2 + " " + equality_sign + " " + arabic_numeral_3);

        answerRoman = convertRomans(answerArabic);
        if(answerRoman.equals("O")) {
          arabicOutput[i] = "Invalid Statement.";
          continue;
        }

        if(arabic_numeral_3 == answerArabic) {
          arabicOutput[i] += " is correct.";
          numberOfCorrectStatement++;
        } else {
             arabicOutput[i] += " is wrong, as the answer should be " + answerRoman + " ,
which is " + answerArabic + " in Arabic numeral form.";
        }
      }

      for(String statement : arabicOutput) {
        System.out.println(statement);
      }
```

```
        System.out.println("\nNumber of Correct Statements = " + numberOfCorrectStatement);

        if(n == 0) n = 1;//to prevent divide by zero error

                    System.out.printf("Percentage of Correct Statements = %2.2f%%", 
((double)numberOfCorrectStatement / n) * 100);

    }

    public static String[] generateInitials(String[] nameArray) {
        String[] initials = new String[nameArray.length];
        for(int i = 0; i < initials.length; i++) {
            nameArray[i] = nameArray[i].trim(); //remove space from the name

             String regex = "[\\s-'_.,]+"; //look for "\s" which means space, - ' _ . , the + sign on the 
back is used to treat consecutive delimiters as a single delimiter

            String[] nameWords = nameArray[i].split(regex);

            for(String nameWord : nameWords) {
                if(nameWord.trim().isEmpty()) {
                    Continue; //To prevent empty space
                }
                 if(nameWord.equalsIgnoreCase("bin") || nameWord.equalsIgnoreCase("binti") || 
nameWord.equalsIgnoreCase("a/l") || nameWord.equalsIgnoreCase("a/p") || 
nameWord.equalsIgnoreCase("al") || nameWord.equalsIgnoreCase("ap") || 
nameWord.equals("@")) {
                    continue;
                }
                if(initials[i] != null) {
                    initials[i] = "" + initials[i] + nameWord.charAt(0);
                }
                else {
                    initials[i] = "" + nameWord.charAt(0);
                }
                initials[i] = initials[i].toUpperCase();
            }
        }
        return initials;
    }


    public static int convertArabics(String roman) {
        int consecutiveLetter = 1;
        char lastRomanLetter, currentRomanLetter, nextRomanLetter;
        int lastArabicLetter, currentArabicLetter, nextArabicLetter;
        int arabicNumber = 0;

        lastRomanLetter = ' ';
        lastArabicLetter = 0;

        int highestArabicLetter = 0;
        int highestArabicIndex = 0;
        int index = 0;

        boolean readFromBehind = false;

        roman = roman.toUpperCase();

        //find the index of the largest roman numeral
        for(int i = 0; i < roman.length(); i++) {
            currentRomanLetter = roman.charAt(i);
            switch(currentRomanLetter) {
                case 'I':
                    currentArabicLetter = 1;
                    break;
                case 'V':
                    currentArabicLetter = 5;
```

```
                    break;
                case 'X':
                    currentArabicLetter = 10;
                    break;
                case 'L':
                    currentArabicLetter = 50;
                    break;
                case 'C':
                    currentArabicLetter = 100;
                    break;
                case 'D':
                    currentArabicLetter = 500;
                    break;
                case 'M':
                    currentArabicLetter = 1000;
                    break;
                default:
                    return 0;
            }
            if(currentArabicLetter > highestArabicLetter) {
                highestArabicIndex = i;
                highestArabicLetter = currentArabicLetter;
            }
        }

            //Check if the largest roman numeral is in front or behind and decide whether it is a
subtraction or addition
        //if readFromBehind is true, it is an subtraction.
        if(highestArabicIndex == 0) {
            index = 0;
            readFromBehind = false;
        }else if(highestArabicIndex == (roman.length() - 1)) {
            index = (roman.length() - 1);
            readFromBehind = true;
        }else{
            return 0;//if the largest letter is in the midlle of the group
        }



        //Loop through every letters
        for(int iteration = 0; iteration < roman.length(); iteration++) {
            currentRomanLetter = roman.charAt(index);

            if(currentRomanLetter == lastRomanLetter) {
                consecutiveLetter++;
                    if(currentRomanLetter == 'V' || currentRomanLetter == 'L' || currentRomanLetter
== 'D') {
                    return 0; //VLD cannot repeated
                }
            }else {
                consecutiveLetter = 1;//reset consecutive letter
            }

            //Determine whether there is nextRomanLetter anymore
            if((iteration + 1) != roman.length()) {
                if(readFromBehind) {
                    nextRomanLetter = roman.charAt(index - 1);
                } else {
                    nextRomanLetter = roman.charAt(index + 1);
                }

            }else {
                nextRomanLetter = ' ';
            }

            switch(currentRomanLetter) {
                case 'I':
                    currentArabicLetter = 1;
        break;
    case 'V':
        currentArabicLetter = 5;
        break;
    case 'X':
        currentArabicLetter = 10;
        break;
    case 'L':
        currentArabicLetter = 50;
        break;
    case 'C':
        currentArabicLetter = 100;
        break;
    case 'D':
        currentArabicLetter = 500;
        break;
    case 'M':
        currentArabicLetter = 1000;
        break;
    default:
        return 0;
}

switch(nextRomanLetter) {
    case 'I':
        nextArabicLetter = 1;
        break;
    case 'V':
        nextArabicLetter = 5;
        break;
    case 'X':
        nextArabicLetter = 10;
        break;
    case 'L':
        nextArabicLetter = 50;
        break;
    case 'C':
        nextArabicLetter = 100;
        break;
    case 'D':
        nextArabicLetter = 500;
        break;
    case 'M':
        nextArabicLetter = 1000;
        break;
    default:
        nextArabicLetter = 0;
}

//If the letter infront is smaller than after
if(lastArabicLetter < nextArabicLetter && iteration != 0) {
    return 0;
}

if(!readFromBehind){
    //if next roman numeral is larger than the current roman numeral
    if(nextArabicLetter > currentArabicLetter) {
        //only IXC can be subtractant
        switch (currentRomanLetter) {
            case 'I':
                if(nextRomanLetter != 'V' && nextRomanLetter != 'X') {
                    return 0;
                }   break;
            case 'X':
                if(nextRomanLetter != 'L' && nextRomanLetter != 'C') {
                    return 0;
                }   break;
            case 'C':
                    if(nextRomanLetter != 'D' && nextRomanLetter != 'M') {
                        return 0;
                    }   break;
                default:
                    return 0;//"0" considered invalid
            }

            //subtract the larger roman numeral from the smaller roman numeral
            arabicNumber = arabicNumber + (nextArabicLetter - currentArabicLetter);

                //update, skip through the second letter as it counted as one, XCV from index 0
skip to index 2
            index++;
            iteration++;

        } else {
            //if current is larger then last, it is seen as invalid except the first iteration
              //Example: MCCIVX --> X is greater than V, so this is an invalid form of roman
numeral system.
            if(lastArabicLetter < currentArabicLetter && iteration != 0) {
                return 0;
            }

            //Add the current value to the total value
            arabicNumber += currentArabicLetter;
        }
    }else{
                //The roman numeral is not in an decreasing order reading from the back.
(According to the sample given in the question)
        if (nextArabicLetter > currentArabicLetter) {
            return 0;
        }

        //Subtract the last roman numeral with the value in front of it
```

```
            if(iteration == 0) {
                arabicNumber = currentArabicLetter; //assign the largest/first roman numeral
value
            }else{
                arabicNumber -= currentArabicLetter; //subtract the largest/first roman numeral
with the following value
            }
        }

        lastRomanLetter = currentRomanLetter;
        lastArabicLetter = currentArabicLetter;

        if(readFromBehind) { index--; }
        else { index++; }

    }

    //Out of range
    if(arabicNumber >= 1 && arabicNumber <= 3999){
        return arabicNumber;
    }else {
        return 0;
    }
  }

  public static String convertRomans(int arabic) {
    if(arabic < 1 && arabic > 3999) {
        return "O";
    }
    String roman = "";
    int digits = 1;
    int arabicLetter;

    while(arabic != 0) {
```

```
arabicLetter = arabic % 10;

String roman_1 = "IIIIVIIIIX";
String roman_2 = "XXXXLXXXXC";
String roman_3 = "CCCCDCCCCM";
String roman_4 = "MMM";

switch(digits) {
  case 1:
    if(arabicLetter < 4 ) {
      roman += roman_1.substring(0, (arabicLetter));
    }else if(arabicLetter == 4 || arabicLetter == 9) {
      roman += roman_1.substring((arabicLetter - 1), (arabicLetter + 1));
    }else {
      roman += roman_1.substring(4, (arabicLetter));
    }
    break;
  case 10:
    if(arabicLetter < 4 ) {
      roman += roman_2.substring(0, (arabicLetter));
    }else if(arabicLetter == 4 || arabicLetter == 9) {
      roman += roman_2.substring((arabicLetter - 1), (arabicLetter + 1));
    }else {
      roman += roman_2.substring(4, (arabicLetter));
    }
    break;
  case 100:
    if(arabicLetter < 4 ) {
      roman += roman_3.substring(0, (arabicLetter));
    }else if(arabicLetter == 4 || arabicLetter == 9) {
      roman += roman_3.substring((arabicLetter - 1), (arabicLetter + 1));
    }else {
      roman += roman_3.substring(4, (arabicLetter));
    }
```

```
                    break;
                case 1000:
                    roman += roman_4.substring(0, (arabicLetter));
                    break;
                default:
                    roman = "O";//invalid code
            }
            roman += " ";

            digits *= 10;
            arabic /= 10;
        }


        //Using an array to prevent reversing issue
        //Example: IVDM reverse it will become
        //MDVI which is wrong, it should be MDIV
        //Reverse the roman number
        String[] romanArray = roman.split(" ");
        for(int i = 0; i < romanArray.length / 2; i++) {
            String temp = romanArray[i];
            romanArray[i] = romanArray[romanArray.length - 1 - i];
            romanArray[romanArray.length - 1 - i] = temp;
        }
        roman = "";
        for(String romanLetter : romanArray) {
            roman += romanLetter;
        }

        return roman;
    }
}
```

Q4 Number Formatter

PROBLEM

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. 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.

SOLUTION

1.Create a function called format that takes an Integer number and an Integer width.

1.1 if length of Integer number > width, return Integer number.

1.2 else, build a for loop to add leading zeros to the Integer number.

1.3 add the Integer number string to the result.

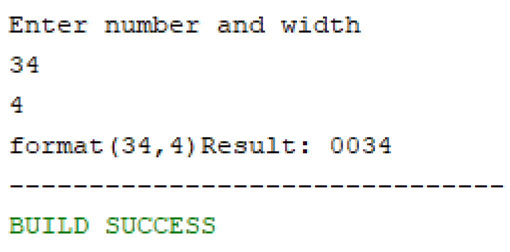
1.4 return the resulting string.

2.Create a main method that prompts user input number and width.

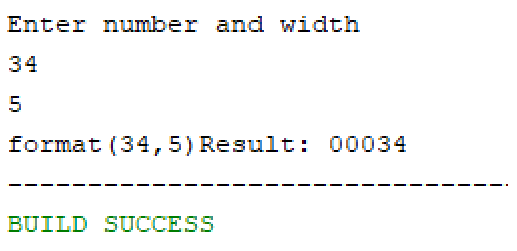
2.1 Calls the function called format to compute and return the resulting string.

2.2 Print the result.

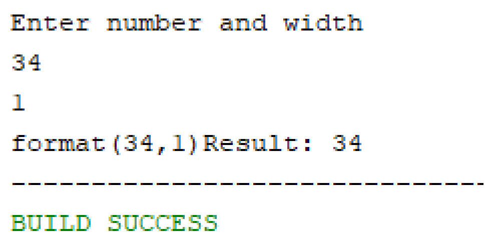
SAMPLE INPUT & OUTPUT



```
Enter number and width
34
4
format(34,4)Result: 0034
--------------------------------
BUILD SUCCESS
```



```
Enter number and width
34
5
format(34,5)Result: 00034
--------------------------------
BUILD SUCCESS
```



```
Enter number and width
34
1
format(34,1)Result: 34
-------------------------------
BUILD SUCCESS
```

SOURCE CODE

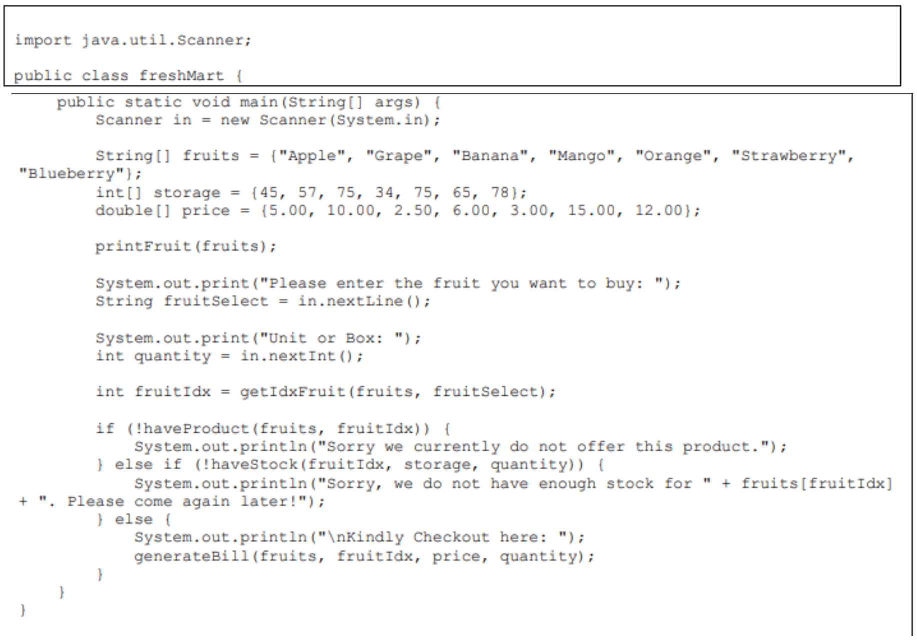
import java.util.Scanner;

```
public class VIVA2Q4 {
    private static String format (int number, int width){
        String str = Integer.toString(number);
        int length = str.length();

        if (length>=width){
            return str;
        }

        String result = "";
        for (int i = 0 ; i < width-length; i++){
            result += "0";
        }

        result += str;
        return result;
    }

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number and width");
        int number = sc.nextInt();
        int width = sc.nextInt();
        System.out.print("format("+number+","+width+")");
        System.out.print("Result: " + format(number,width));
    }
}
```

Q5 Fruit Store Billing System

PROBLEM

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 pri

SOLUTION

In printFruit(String[] fruits) method:

1. Use a for loop to store the different types of fruit in the fruits array.
2. Print out fruits listed in the array.

In getIdxFruit(String[] fruits, String fruitSelect) method:

1. Use a for loop to loop through the index of fruits in the String[] fruits.
2. Use If-else condition to check if fruitSelect is equal to fruits[i]
3. Use compareToIgnoreCase to allow comparison with fruitSelect by ignoring uppercase and lowercase letters
4. Return index fruit if true and -1 if false.

In haveProduct(String[] fruits, int fruitIdx) method:

1. Use a for loop to loop through the numbers of the fruits.
2. Use a If-else condition to check if the index of the fruit is equal to the index of fruit in the fruits array.
3. equalsIgnoreCase used to ignore the uppercase and lowercase letters.
4. Return true if if statement is correct and false if incorrect.

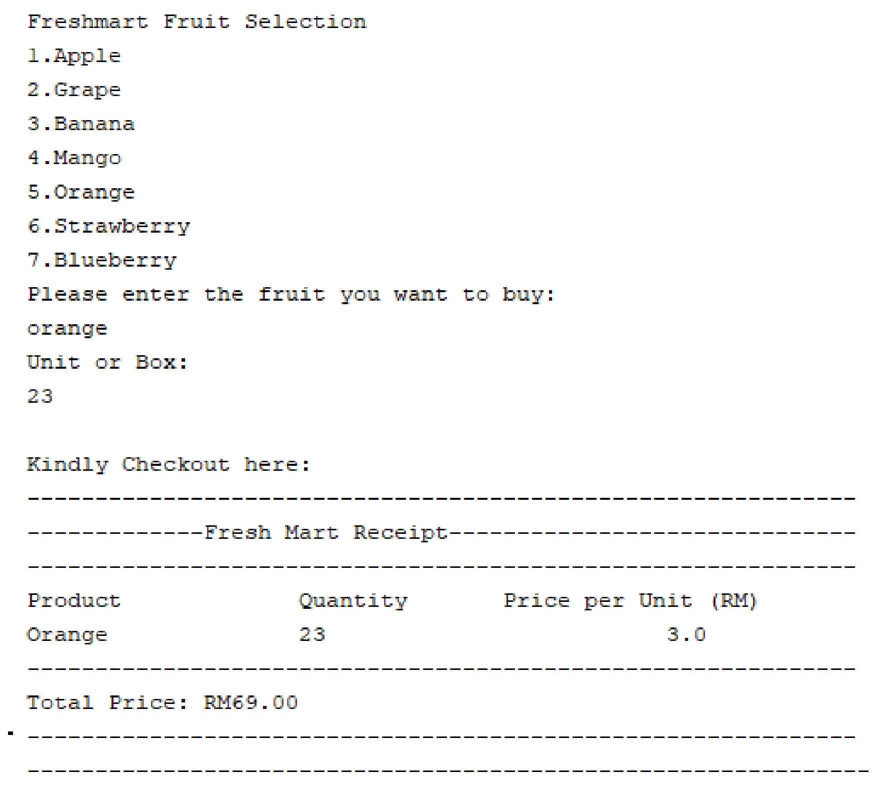
In haveStock(int fruitIdx, int[] storage, int quantity) method:

1. Return true if the quantity of the fruitsSelected are less or equal to the storage of the fruit index.
2. Return false if quantity of the fruitSelected is more than the storage of the fruit index.

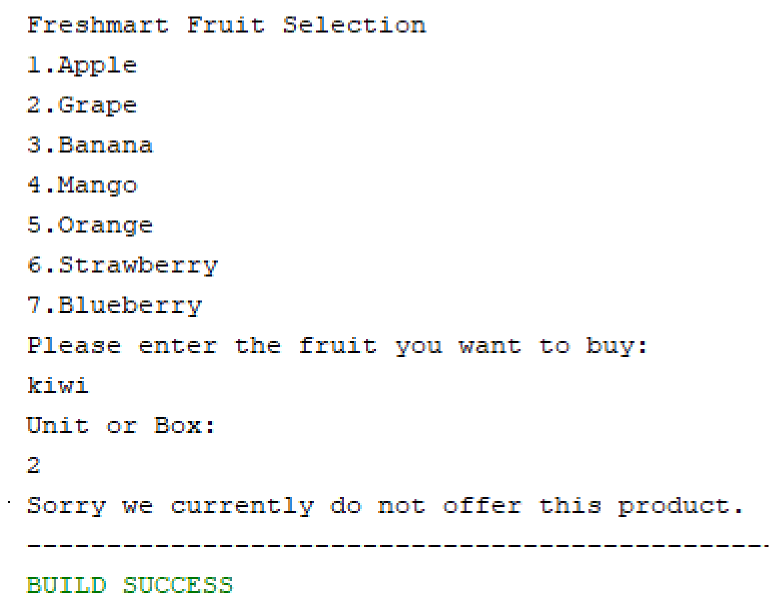
In generateBill(String[] fruits, int fruitIdx, double[] price) method:

1. Print formatted receipt based on the question.
2. Use a formatted output (System.out.printf)) to allow alignment of spaces.
   E.g (“%-15s”, “Product”) : 15 spaces are aligned to the left, “Product” which takes up 7 spaces to the left, remaining 8 blank spaces to fill up.
3. Print the type of fruits, its quantity (the user input) and the price of the fruit.
4. Calculate the total price of the fruit based on its quantity.
5. Print the total price of fruit.

SAMPLE INPUT & 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                         3.0
-------------------------------------------------------------
Total Price: RM69.00
-------------------------------------------------------------
--------------------------------------------------------------
```



```
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:
kiwi
Unit or Box:
2
Sorry we currently do not offer this product.
------------------------------------------------
BUILD SUCCESS
```

SOURCE CODE

```
import java.util.Scanner;
public class VIVA2Q5 {
    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.println("Please enter the fruit you want to buy: ");
 String fruitSelect = in.nextLine();
 System.out.println("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);
 }
 }

 private static void printFruit (String [] fruits){
    System.out.println("Freshmart Fruit Selection");
    for (int i = 0; i < fruits.length; i ++){
    System.out.println( (i+1) + "." + fruits[i]);
 }
 }
```

```
private static int getIdxFruit (String[]fruits, String fruitselect){
        for (int i = 0 ; i<fruits.length; i ++) {
            int result = fruits[i].compareToIgnoreCase(fruitselect);
            if (result == 0)
                return i;
        }
    return -1;
    }

private static boolean haveProduct(String[] fruits, int fruitIdx){
return fruitIdx >=0;
}

private static boolean haveStock(int fruitIdx, int[] storage, int quantity){
    return quantity <= storage[fruitIdx];
}

private static void generateBill(String[] fruits, int fruitIdx, double[] price, int quantity){
     System.out.println("-------------------------------------------------------------");
     System.out.println("-------------Fresh Mart Receipt-------------------------------");
     System.out.println("-------------------------------------------------------------");
     System.out.printf("%-20s%-15s%15s\n","Product ", "Quantity ", "Price per Unit (RM)");
     System.out.printf("%-20s%-15s%15s\n",fruits[fruitIdx],  quantity , price[fruitIdx]);
     System.out.println("-------------------------------------------------------------");
     System.out.printf("Total Price: RM");
     System.out.printf("%.2f\n",quantity * price[fruitIdx]);
     System.out.println("-------------------------------------------------------------");

}
}
```

Q6 Matrix Manipulation

PROBLEM

Write code that

1. Take input of n x n integers to form a matrix.
2. Rotate the matrix 90 degrees clockwise.
3. Display the original matrix and the rotated matrix.

SOLUTION

In main method:

1. Initialize an 2D array called matrix by calling parseMatrix method.
2. Call displayMatrix method to print the original matrix.
3. Initialize another 2D array called rotatedMatrix by calling rotateMatrix method with argument 'matrix' to rotate the matrix.
4. Call displayMatrix method to print the rotated matrix.

In parseMatrix method:

1. Create a Scanner object.
2. Declare an integer variable n for user to input the dimension of the matrix.
3. Declare an integer 2D array called matrix to represent the matrix.
4. Let user to input the number of matrix by using 2 for loops. The first for loop is to repeat the row and the second for loop is to repeat the column.
5. Return the matrix to main method.

In displayMatrix method:

1. Print the matrix by using two for loops.

In rotateMatrix method:

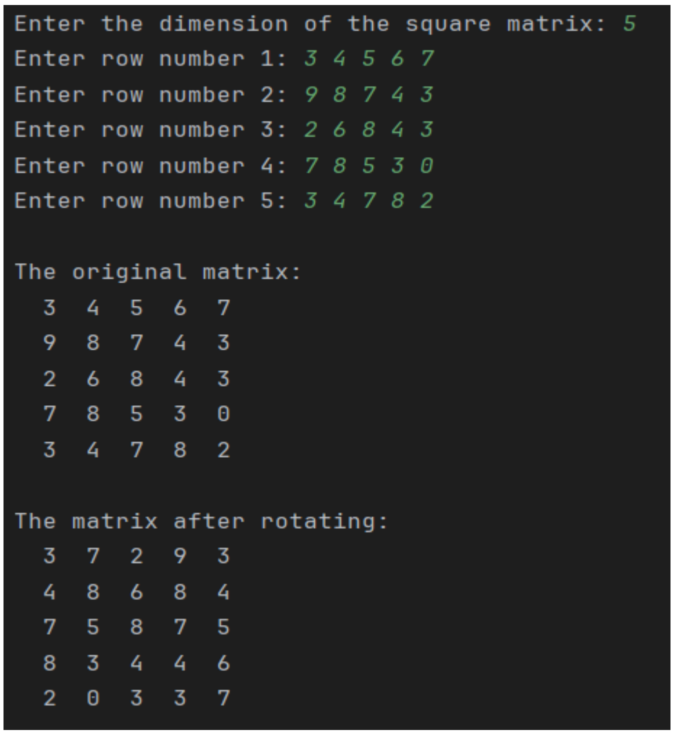
1. Initialize an integer n to represent the matrix length for future use.
2. Declare a 2D array called rotated matrix.

3. First, inverse the position of each integer in matrix.

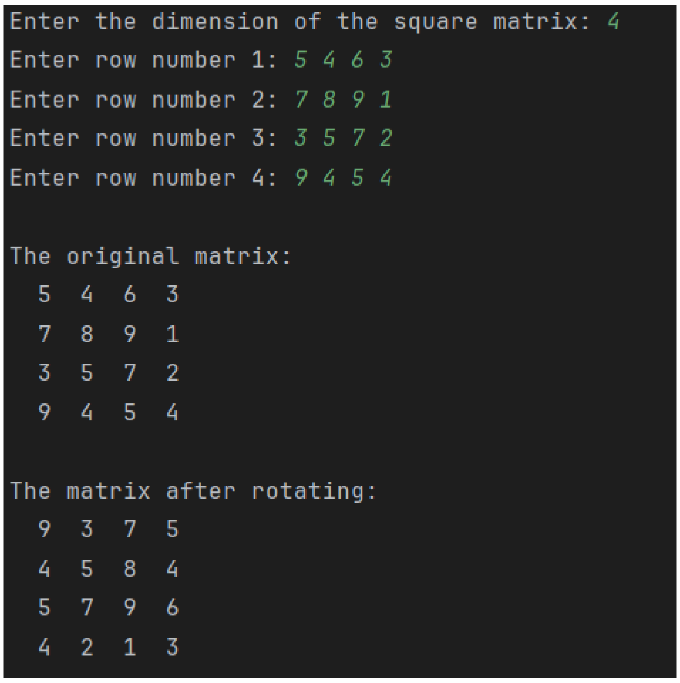
   (Eg: integer in position(i,j) change to position (j,i))
4. Second, inverse the position of integer for each row in the matrix.

   (Eg: 1 2 3 → 3 2 1)

<u>SAMPLE INPUT & OUTPUT</u>



```
Enter the dimension of the square matrix: 5
Enter row number 1: 3 4 5 6 7
Enter row number 2: 9 8 7 4 3
Enter row number 3: 2 6 8 4 3
Enter row number 4: 7 8 5 3 0
Enter row number 5: 3 4 7 8 2

The original matrix:
  3  4  5  6  7
  9  8  7  4  3
  2  6  8  4  3
  7  8  5  3  0
  3  4  7  8  2

The matrix after rotating:
  3  7  2  9  3
  4  8  6  8  4
  7  5  8  7  5
  8  3  4  4  6
  2  0  3  3  7
```



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

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

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

SOURCE CODE

```
import java.util.Scanner;
public class V2Q6 {
   public static void main(String[] args) {
      int[][] matrix = parseMatrix();
      System.out.println("The original matrix:");
      displayMatrix(matrix);
      System.out.println("The matrix after rotating:");
      int[][] rotatedMatrix = rotateMatrix(matrix);
      displayMatrix(rotatedMatrix);
   }

   public static int[][] parseMatrix() {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter the dimension of the square matrix: ");
      int n = sc.nextInt();
      int matrix [][] = new int [n][n];
      for (int i = 0; i<n; i++) {
         System.out.print("Enter row number " + (i+1) + ": ");
         for (int j=0; j<n; j++) {
            matrix[i][j] = sc.nextInt();
         }
      }
      System.out.println();
      sc.close();
      return matrix;
   }

   public static void displayMatrix(int [][]matrix) {
      for (int i=0; i<matrix.length; i++) {
         for (int j=0; j<matrix[i].length; j++) {
            System.out.printf("%3d", matrix[i][j]);
         }
         System.out.println();
```

```
        }
        System.out.println();
    }

    public static int[][] rotateMatrix (int [][]matrix) {
        int n = matrix.length;
        int[][] rotatedMatrix = new int[n][n];
        for (int i = 0; i < n ; i++) {
            for (int j = 0; j < n ; j++) {
                rotatedMatrix[i][j] = matrix[n - 1 - j][i];
            }
        }
        return rotatedMatrix;
    }

}
```