Data Structures and Algorithms 1 Assignment

Name: Deborah Vella

Id number: 366299(M)

Course: Artificial Intelligence

Course code: ICS1018

Table of Contents

Question 1: Write a program that, given a list of integers, finds all 2-pairs of integers that have the same product. A 2-pair is 2 distinct pairs of integers ((a,b),(c,d)) where a X b = c X d and a ≠ b ≠ c ≠ d. The range of integers in the list should be from 1 to 1024.

Statement of completion: Attempted and completed

The user has to enter how many integers he wants to input. The program validates his input and makes sure not to choose a number less than 4, because with such a small list no pairs can be found. This is an optimisation to save time from searching for pairs when they clearly can never be found. When the list is entered, the algorithm validates that input as well and makes sure that the numbers are not greater than 1024. The method pairs() uses nested for loops to find all products that match and have different factors.

Testing

prompt	Input	Expected Output	Actual Output
Ask how many numbers are to be entered	10	You can start entering your numbers	You can start entering your numbers
Ask how many numbers are to be entered	3	You need to enter more than 3 integers to find pairs	
Asked to enter numbers	2 6 7 8 4 3 16 1024 1 512	Outputs all distinct pairs and number of pairs found	Outputs all distinct pairs and number of pairs found
User enters a number larger than 1024	2666	Please enter an integer that is less than or equal to 1024	Please enter an integer that is less than or equal to 1024
User enters numbers that result in no pairs	1 2 9 4	No pairs were found	No pairs were found

Output:

10 numbers are entered and all distinct pairs are outputted.

```
How many numbers do you want to enter?
10
You can start entering your numbers:
2
6
7
8
4
3
16
1024
1
512
Distinct pair of integers : ((2,6),( 4,3))
Distinct pair of integers : ((2,8),( 16,1))
Distinct pair of integers : ((2,4),( 8,1))
Distinct pair of integers : ((2,3),( 6,1))
Distinct pair of integers : ((2,16),( 8,4))
Distinct pair of integers : ((2,1024),( 4,512))
Distinct pair of integers : ((2,512),( 1024,1))
Distinct pair of integers : ((6,8),( 3,16))
Distinct pair of integers : ((6,4),( 8,3))
Distinct pair of integers : ((6,512),( 3,1024))
Distinct pair of integers : ((8,1024),( 16,512))
Distinct pair of integers : ((8,512),( 4,1024))
A total of distinct pairs:12
```

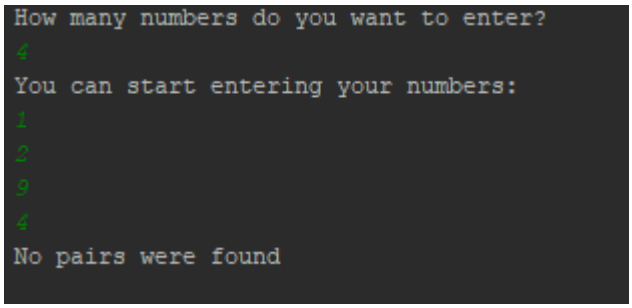
When the user wants to enter less than 4 numbers the program does not let him as there can never be pairs for a list which is less than 4.

```
How many numbers do you want to enter?
3
You need to enter more than 3 integers to be able to find pairs.
```

When the user enters a number greater than 1024, he is asked to re-enter integer until it's a valid one.

```
2666
Please enter an integer that is less than or equal to 1024
99907
Please enter an integer that is less than or equal to 1024
77
```

When no pairs are found



Source Code:

question_1.java

```
import java.util.*;
public class question_1 {

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

        System.out.println("How many numbers do you want to enter?");
        int n = input.nextInt();
        while(n<=3){ //optimisation because when the list is less than 4, no
distinct pairs can be found
            System.out.println("You need to enter more than 3 integers to be able
to find pairs.");
            n = input.nextInt();
        }

        System.out.println("You can start entering your numbers: ");
        int[] array = new int[n];  //declare array with the entered amount as size

        for(int i=0;i<n;i++)
        {
                array[i] = input.nextInt();
                while(array[i] > 1024){
                    System.out.println("Please enter an integer that is less than
or equal to 1024");
                    array[i] = input.nextInt();
                }
        }

        pairs(array, n); //call method pairs

    }

    static void pairs (int [] array, int n){
        int product_1 =0;
        int product_2 =0;
        int counter=0;

        //counters of for loops
        int h;
        int i;
        int j;
        int k;

        //for loop gets the 1st pair's first number. size/2 for optimization
        for(h =0; h< n/2;h++){
            //for loop gets the 1st pair's second number.
            for(i=h;i<n;i++){
                product_1 = array [h]*array[i];
                //for loop gets the 2nd pair's first number. Starting from element
after that of the first pair
                for (j = h+1 ; j<n;j++){
                    //for loop gets the 2nd pair's second number.
                    for (k=j; k<n;k++){
                        product_2= array[j]*array[k];
                        //comparing the prodcuts results
                        if(product_1==product_2){
      the following if statements make sure that none of the integers are the same
            if(array[h]!=array[i] && array[h] != array[j] && array[h] !=array[k]) {
                        if(array[i] != array[j] && array[i] !=array[k]) {
                                    if(array[j]!=array[k]) {
        //  if pair is found without any repeated integers it is printed
System.out.println("Distinct pair of integers : ((" + array[h] + "," + array[i] +
"),( " + array[j] + "," + array[k]+ "))");
 counter++; // number of distinct pairs found is incremented
```

```
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        if(counter>0){ //if counter is greater than 0 display number of pairs found
            System.out.println("A total of distinct pairs:" + counter);

        }else { //else notify the user that no pairs were found
            System.out.println("No pairs were found");
        }
    }
}
```

Question 2: Write a program that uses an ADT Stack to evaluate arithmetic expressions in RPN format. The contents of the stack should be displayed on the screen during evaluation. The allowed arithmetic operators are +, -, x, and /

Statement of completion: Attempted and completed

For this question the stack functions push() and pop() are used to populate or remove from top of stack. A switch case is used so that when an operator is used the program pops two times and performs the respective calculation and pushes the answer on top of stack again.

Output:

This is an example that uses all four operators. RPN used is: 5 5 + 2 8 3 - * /

```
Please specify the length of you RPN (numbers and operators):
9
The RPN should be entered number/operator at a time (starting from the left most side of your expression)
Enter number/operator:
5
Push        [5.0]
Enter number/operator:
5
Push        [5.0, 5.0]
Enter number/operator:
+
Performing addition...      [10.0]
Enter number/operator:
2
Push        [10.0, 2.0]
Enter number/operator:
8
Push        [10.0, 2.0, 8.0]
Enter number/operator:
3
Push        [10.0, 2.0, 8.0, 3.0]
Enter number/operator:
-
Performing subtraction...       [10.0, 2.0, 5.0]
Enter number/operator:
*
Performing multiplication...        [10.0, 10.0]
Enter number/operator:
/
Performing division...      [1.0]
Result is: 1.0
```

Source Code

question_2.java

```
import java.util.*;

public  class question_2 {

    public static void main(String[] args) {
        String input;

        Stack<Double> stack = new Stack<>();
        Scanner sc = new Scanner(System.in);

        System.out.println("Please specify the length of you RPN (numbers and
operators): ");
        int length = sc.nextInt();

        System.out.println("The RPN should be entered number/operator at a time
(starting from the left most side of your expression)");

        for (int i = 0; i < length; i++) {     //loops as long as i is less than
the length of the RPN expression
            System.out.println("Enter number/operator: ");
            input = sc.next();

            double no1 = 0.0;    //these variables will later store the popped
values
            double no2 = 0.0;

            switch (input) { //the switch checks if the input is an operator and
performs corresponding calculations
            //calculations are done by popping twice and performing the operation
on the two popped values
                case "+":
                    System.out.print("Performing addition...\t\t");

                    no2 = stack.pop();
                    no1 = stack.pop();

                    stack.push(no1 + no2);
                    break;
                case "-":
                    System.out.print("Performing subtraction...\t\t");

                    no2 = stack.pop();
                    no1 = stack.pop();

                    stack.push(no1 - no2);
                    break;
                case "/":
                    System.out.print("Performing division...\t\t");

                    no2 = stack.pop();
                    no1 = stack.pop();

                    stack.push(no1 / no2);
                    break;
                case "*":
                    System.out.print("Performing multiplication...\t\t");

                    no2 = stack.pop();
                    no1 = stack.pop();

                    stack.push(no1 * no2);
                    break;

                default:    //if input is not an operator then push the number and
```

```
store it as double
                    System.out.print("Push\t\t");
                    stack.push(Double.parseDouble(input));
                    break;
            }

            System.out.println(stack); //prints content of stack
        }

        System.out.println("Result is: " + stack.pop());  //output answer of rpn

    }
}
```

Question 3: Write a Boolean function that checks if a number is prime. Also implement the Sieve of Eratosthenes algorithm. Explain any optimizations made.

Statement of completion: Attempted and completed

The program has two functions, one for the prime numbers and the other for the sieve of Eratosthenes. As an optimisation in finding whether it is prime or not, the square root of the entered number is found so that the program will search from 2 up to the square root. Then it checks if entered number is divisible by two, and if it is not, the algorithm searches only through the odd numbers.

Testing

prompt	Input	Expected Output	Actual Output
Check if prime number			
Enter a number	19	19.0 is a prime	19.0 is a prime
Enter a number	200	200.0 is not a prime number.	200.0 is not a prime number.
Sieve of Eratosthenes			
The limit is asked to be entered	30	2 3 5 7 11 13 17 19 23 29	2 3 5 7 11 13 17 19 23 29

Output:

Case 1: Prime number is entered

```
Enter a number:
19
19.0 is a prime number.
```

Case 2: A number that is not prime is entered

```
Enter a number:
200
200.0 is not a prime number.
```

Sieve of Eratosthenes

```
Enter the limit up until the Sieve of Eratosthenes is to be computed (enter an integer):
30
2
3
5
7
11
13
17
19
23
29
```

Source Code

Question_3.java

```
import java.util.*;
import java.lang.Math;

public class question_3 {
    public static void main(String[] args) {
        double num;
        boolean k;
        int limit;

        Scanner input = new Scanner(System.in);

        System.out.println("Enter a number: ");
        num = input.nextInt();
        k = prime(num);
        if(!k) //if the method prime() returns false, the number isn't prime,  else
it is prime
        {
            System.out.println(num + " is not a prime number.");
        }
        else{
            System.out.println(num+ " is a prime number.");
        }

        //Sieve of Eratosthenes
        System.out.println("Enter the limit up until the Sieve of Eratosthenes is
to be computed (enter an integer): ");
        limit = input.nextInt();
        sieve_of_Eratosthenes(limit);

    }

    private static boolean prime(double num)
    {
        double root;
        double rounded;
        double mod;
        int i;
        int j=0;

        root= Math.sqrt(num);       //calculate square root of the number
        rounded = Math.round(root); //round the square root in case it isn't an
integer

        mod = num % 2;      //checking if the number is divisible by 2

        if(mod == 0)        //if the remainder is 0 it means that it is not a prime
number
        {
            if(num == 2)        //caters for the case when the number is 2
            {
                return true;
            }else {
                return  false;
            }
        }
        else{ //if the remainder isn't equal to 0, the program loops up until the
square root of the number and checks only the odd numbers

            for(i=3; i<=rounded; i++)
            {
                mod = num % i;
                i++;

                if(mod == 0)  //if the number is found to not be prime set j equal
```

```
to one and break the loop
                {
                    j = 1;
                    break;
                }
            }

            if(j == 1)      //if j is 1 it means that it was found to not be prime
else it is prime
            {
                return false;
            }
            else{
                return true;
            }
        }

    }


    private static void sieve_of_Eratosthenes(int limit){
        int array[] = new int [limit];
        int h;
        for(int i=0; i<limit; i++){
            array[i] = i+1;  //populating the array
        }
         int n = 2;     //starting from n is 2
        while(n<=limit){
            h=1;
            while((n*h)<limit){ //while multiple is smaller than the size of the
list
                h++;
                for(int j=0; j<limit; j++){ //loops through the list
                    if(array[j] == (n*h) || array[j] == 1){
                        array[j]=0; //setting to zero if equal to the previously
found multiple
                    }
                }
            }
            n++; //moving to the next prime number
        }

        //prints out all the prime numbers found
        for(int j=0; j<limit; j++){
            if(array[j]!=0){ //prints only the elements which are not equal to
zero, therefore, they are prime
                System.out.println(array[j]);
            }
        }

    }
}
```

Question 4: Write a program that accepts an input a sequence of integers (one by one) and then incrementally builds a Binary Search Tree (BST). There is no need to balance the BST.

Statement of completion: Not attempted

Question 5: Write a program that finds an approximation to the square root of a given number n using an iterative numerical method such as the Newton Raphson Method

Statement of completion: Attempted and completed. The program works on integers only and no decimal numbers.

To calculate the square root the Babylonian method was implemented. An initial guess is made, a formula is then applied to get a more accurate guess. The new estimation and previous estimation are compared and this algorithm keeps on looping until both estimations match up to 4 decimal places.

Testing:

prompt	Input	Expected Output	Actual Output
Enter a number to apply square root on	144	Square root of 144 is: 12	Square root of 144.0 is: 12.0
Enter a number to apply square root on	8	Square root of 8 is: 2.8284	Square root of 8 is: 2.8284

Output:

Case 1: A square number is entered

```
Enter a number to apply square root on:
144
Square root of 144.0 is: 12.0
```

Case 2: A number which isn't a perfect square number

```
Enter a number to apply square root on:
8
Square root of 8.0 is: 2.8284
```

Source Code

question_5.java

```
import java.util.*;
public class question_5 {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        double num;     //stores the user input
        double new_est; //stores the last estimation found8
        double prev_est;//stores the previous estimation
        double x1;      //stores the answer when the formula of the babylonian
method is computed
        double xn;      //stores the last estimation
        int toRound;    //stores an integer value of the newest estimation
        int toRound2;   //stores an integer value of the previous estimaion
        double answer;  //stores final answer

        //user is asked to enter a number
        System.out.println("Enter a number to apply square root on: ");
        num = input.nextInt();  //storing the input inside the variable num

        //divide the entered number by 2 to find the first estimation
        xn = num/2;

        do{

          x1 = (xn + (num/xn))/2;   //This formula is computed to find a more
accurate estimation of the root and is stored inside x1.
          new_est = x1;             //new_est and prev_est are set equal to x1 an
xn so that further working is made on new_est and prev_est while x1 and xn
          prev_est = xn;            //remain unchanged
          xn = x1;                  //set xn equal to the last estimation so that
the new value is used in the next iteration

            toRound = (int)(new_est * 10000.0);     //typecasting new_est and
prev_est from double to int.  The value is multipled by 10,000 so that
            toRound2 = (int)(prev_est * 10000.0);   //we can compare the numbers up
to the 4th decimal place

        }while(toRound != toRound2 ); //continues looping until toRound and
toRound2 are equal to each other.  This means that the last estimation and
        //the previous estimation matches up until the 4th decimal place

        answer = toRound/10000.0; //the value inside toRound is divide by 10,000 to
get back the original decimal value.  This is the final answer

        System.out.println("Square root of "+num+" is: "+answer); //outputs the
answer on screen
    }
}
```

Question 6: Write a program that, given an array of integers, finds all integers in the array that are repeated more than once. Try to find the fastest and most memory-efficient way of doing this.

Statement of completion: Attempted and completed

At first, I implemented a different algorithm where each number stored in the array referenced a different index of the array. If two numbers referenced the same index, it implied that that number is repeated. But this algorithm worked only on numbers that are less than the length of the list. Therefore, I chose the algorithm shown below so that the program works on all numbers.

Testing:

prompt	Input	Expected Output	Actual Output
Enter amount of numbers to be entered	8	Prompt the user 8 times to input a number	It Prompts the user 8 times to input a number
All 8 numbers entered with duplicates in the list	1 55 999 28 55 6 999 1	The repeated integers are: 1 55 999	The repeated integers are: 1 55 999
All 8 numbers are entered and numbers are repeated more than twice	7 11 11 11 8 9 9 6	Checking for repeated integers… Repeated: 11 Repeated: 11 Repeated:9	Checking for repeated integers… Repeated: 11 Repeated: 11 Repeated:9
None of the entered integers are repeated	1 2 3	Checking for repeated integers… There are no repeated integers	Checking for repeated integers… There are no repeated integers

Output:

Case 1: Duplicates

```
Please enter the amount of numbers to be entered:
8
You can start inputting the integers
Enter:
1
Enter:
55
Enter:
999
Enter:
28
Enter:
55
Enter:
6
Enter:
999
Enter:
1
Checking for repeated integers...
repeated:
1
repeated:
55
repeated:
999
```

Case 2: Repeated more than twice. Here the program does not print the repeated more than twice integers, once.

```
Please enter the amount of numbers to be entered:
7
You can start inputting the integers
Enter:
11
Enter:
11
Enter:
11
Enter:
8
Enter:
9
Enter:
9
Enter:
6
Checking for repeated integers...
repeated:
9
repeated:
11
repeated:
11
```

If there are no repeated integers the user is notified.

```
Please enter the amount of numbers to be entered:
3
You can start inputting the integers
Enter:
1
Enter:
2
Enter:
3
Checking for repeated integers...
There are no repeated integers.
```

Source code

question_6.java

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

public class question_6 {

    public static void main(String[] args)
    {

        Scanner input = new Scanner(System.in);

        //asking the user to input the length of the integer list and it is then
set equal to the size of the array to be used
        System.out.println("Please enter the amount of numbers to be entered: ");
        int size = input.nextInt();
        int []array = new int[size];  //declaring the array with the user defined
size
        int flag = 0;

        System.out.println("You can start inputting the integers");
        for(int i=0; i<size; i++){  //loops through the whole array and populates
it after each iteration
            System.out.println("Enter: ");
            array[i] = input.nextInt();
        }

        System.out.println("Checking for repeated integers... ");
        Arrays.sort(array); //sorts out the array

        for(int j = 1; j < size; j++) {  //loops through the sorted array
            if(array[j] == array[j - 1]) {  //compares the current element with the
previous one
                System.out.println("repeated: ");
                System.out.println(array[j]); //if the two elements compared are
equal, then output the number
                flag=1;
            }
        }

        if(flag==0){
            System.out.println("There are no repeated integers.");
        }
    }
}
```

Question 7: Write a recursive function that finds the largest number in a given list of integers.

Statement of completion: Attempted and completed.

Testing:

prompt	Input	Expected Output	Actual Output
Enter amount of numbers to be entered	9	Prompt the user 9times to input a number	It Prompts the user 9 times to input a number
All 9 numbers entered with duplicates in the list	700 5 1 88 9 1000 8 50 10	The maximum number in your list is: 1000	The maximum number in your list is: 1000

Output:

```
How many numbers do you want to enter?
9
You can start entering your numbers:
Enter number:
700
Enter number:
5
Enter number:
1
Enter number:
88
Enter number:
9
Enter number:
1000
Enter number:
8
Enter number:
50
Enter number:
10
The maximum number in your list is: 1000
```

Source code

question_7.java

```
import java.util.*;

public class question_7 {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n; //size of list
        int index2 = 0;
        int max=0;

        System.out.println("How many numbers do you want to enter?");
        n = input.nextInt();
        System.out.println("You can start entering your numbers: ");
        int[] a = new int[n];   //declare array with the entered size

        for(int i=0; i<n; i++)  //populate the array with user input
        {
            System.out.println("Enter number: ");
            a[i]=input.nextInt();
        }

        int maximum = greatest(a, index2, n, max);  //store the returned integer
from greatest()
        System.out.println("The maximum number in your list is: "+maximum);
    }

    private static int greatest(int[] a, int index2,int n, int max) {

        if (index2==n)  //if all elements in the array are checked return maximum
        {
            return max;
        }else{
            if(max>a[index2]) //if last found max is greater than the element found
inside the index pointed by index2, call the function again passing same max and
increase the array index
            {
                max = max;
                return greatest(a,index2+1, n,max);
            }
            else{
                max = a[index2]; //else the maximum is updated to the new one found
a position indicated by index2
                return greatest(a, index2+1, n,max); //call function again
            }
        }
    }
}
```

Question 8: Write a function that computes cosine or sine by taking the first n terms of the appropriate series expansion.

Statement of completion: Attempted and completed.

In this question the Maclaurin Series expansions of sine and cosine were implemented. The user can choose which one he wants to use at the beginning of the program and after one of them is calculated.

Testing:

prompt	Input	Expected Output	Actual Output
Choose between sine or cosine or exit	1	Prompt the user to enter the number to calculate the sine of	Enter a number to calculate the sine of:
Enter a number to calculate the sine of:	3.14	Prompt the user to enter a limit of the series	Enter a number that represent the first n terms:
Asked to enter limit of series	6	sin(3.14) result	The answer for sine of 3.14 is: 0.0026434192764328446
Choose between sine or cosine or exit	2	Prompt the user to enter the number to calculate the cosine of	Enter a number to calculate the cosine of:
Enter a number to calculate the cosine of:	3.14	Prompt the user to enter a limit of the series	Enter a number that represent the first n terms:
Asked to enter limit of series	6	cos(3.14) result	The answer for cosine of 3.14 is: -0.9998989681026615
Choose between sine or cosine or exit	3	Terminate program	Exiting… Program terminates

Output:

Sine is chosen and sin(3.14) is calculated according to the series limit specified during execution

```
----Menu----
1. Sine
2. Cosine
3.Exit
1
Enter a number to calculate the sine of:
3.14
Enter a number that represent the first n terms:
6
The answer for sine of 3.14 is: 0.0026434192764328446
----Menu----
1. Sine
2. Cosine
3.Exit
```

Cosine is chosen and cos(3.14) is calculated according to the series limit specified during execution

```
----Menu----
1. Sine
2. Cosine
3.Exit
2
Enter a number to calculate the cosine of:
3.14
Enter a number that represent the first n terms:
6
The answer for cosine of 3.14 is: -0.9998989681026615
----Menu----
1. Sine
2. Cosine
3.Exit
```

Exit is chosen

```
----Menu----
1. Sine
2. Cosine
3.Exit
3
Exiting....
```

Source Code:

question_8.java

```
import java.util.Scanner;

public class question_8 {
    public question_8() {
    }

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

        int choose;
        do {
            System.out.println("----Menu----");
            System.out.println("1. Sine\n2. Cosine\n3.Exit");
            choose = input.nextInt();
            switch(choose) {
                case 1:
                    sine();
                    break;
                case 2:
                    cosine();
                    break;
                case 3:
                    System.out.println("Exiting....");
                    break;
                default:
                    System.out.println("Wrong input!");
            }
        } while(choose != 3);

    }

//performs the factorial
    public static int fact(int n) {
        if (n <= 1)
            return 1;
        else
            return n * fact(n - 1);
    }

    public static void cosine() {
        double answer = 0.00;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a number to calculate the cosine of: ");
        double x = input.nextDouble();
        System.out.println("Enter a number that represent the first n terms: ");
        double n = (double)input.nextInt();

        //performs the maclaurin expansion calculation for cosine
        for(int i = 0; (double)i <= n; ++i) {
            double power1 = Math.pow(-1.0D, (double)i);
            int times = 2 * i;
            int factorial = fact(times);
            double power2 = Math.pow(x, (double)times);
            double div = power2 / (double)factorial;
            double cosine = power1 * div;
            if (i == 0) {
                answer = 1.0D;
            } else {
                answer += cosine;
            }
        }
```

```
        System.out.println("The answer for cosine of " + x + " is: " + answer);
    }

    public static void sine() {
        double answer = 0.00;
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a number to calculate the sine of: ");
        double x = input.nextDouble();
        System.out.println("Enter a number that represent the first n terms: ");
        double n = (double)input.nextInt();

        //performs the maclaurin expansion calculation for sine
        for(int i = 0; (double)i <= n; ++i) {
            double power1 = Math.pow(-1.0D, (double)i);
            int times = 2 * i + 1;
            int factorial = fact(times);
            double power2 = Math.pow(x, (double)times);
            double div = power2 / (double)factorial;
            double sine = power1 * div;
            if (i == 0) {
                answer = x;
            } else {
                answer += sine;
            }
        }

        System.out.println("The answer for sine of " + x + " is: " + answer);
    }
}
```

Question 9: Write a function that returns the sum of the first n numbers of the Fibonacci sequence (Wikipedia).

Statement of completion: Attempted and Completed

For this task, a for loop was used to generate a Fibonacci sequence and find the total sum between the numbers generated.

Testing:

prompt	Input	Expected Output	Actual Output
Enter amount of numbers to be entered	8	Outputs the computed Fibonacci sequence up to 8 terms and their sum	Fibonacci Sequence of the first 8 terms: 1 2 3 5 8 13 21 Total sum: 54

Output:

```
Enter the number of the first n terms:
8
Fibonacci Sequence of the first 8 terms:
1
2
3
5
8
13
21
Total sum: 54
```

Source Code

question_9.java

```
import java.util.*;

public class question_9 {
    public static void main(String[] args) {

        int num;     //stores the first n number of the series
        int i;       //used as a counter
        int a = 0;   //initialised to 0
        int b = 1;   //initialised to 1
        int c;       //stores  the number obtained by adding the two previous
numbers
        int sum=0;   //stores the final sum of the first n numbers

        Scanner input = new Scanner(System.in);

        System.out.println("Enter the number of the first n terms: ");
        num = input.nextInt();

        System.out.println("Fibonacci Sequence of the first " +num+ " terms:");
        //starts from the second number (i.e 1) in the Fibonacci series
        for (i = 1; i < num; i++) {
            c=a+b;       //set c equal to the sum of the two previous numbers in the
series
            a=b;         //update a with the elements in b
            b=c;         //update b with the element in c which is the last number
found in the series
            sum = sum+c; //add the last found sum to the new number stored in c
            System.out.println(c);
        }

        //After each iteration the variables a and b store the last two numbers of
the series so far.  The variable c stores the last number and sum stores all the
added terms so far.

        sum=sum+1; //caters for the first number (1) in the fibonacci series

        System.out.println("Total sum: "+ sum);
    }
}
```