NATIONAL UNIVERSITY
551 M.F. JHOCSON ST., SAMPALOC, MANILA

COMPUTER ENGINEERING
CPOOPG2L: Object Oriented Programming

NATIONAL UNIVERSITY
COLLEGE OF ENGINEERING

Name/Section	Seth Marcus Martin COE211	Rating
Date performed	December 14, 2022	
Date submitted	December 15, 2022	
Professor	Eng. Marlon Bagara	

EXPERIMENT NO. 3
REPETITION STATEMENTS (the loop condition)

I. OBJECTIVES:

This activity aims to:

1. Recognize the different forms of executing repetitive tasks.
2. Familiarize the procedures of executing repetitive task.

LEARNING OUTCOMES (LOs) *At the end of the activity, the students should be able to:*	COURSE LEARNING OUTCOMES (CLOs)				
	1	2	3	4	5
1. Write a program that will implement the for, while and do while loop statement.		•	•		

COURSE LEARNING OUTCOMES (CLOs)

1. Understand the fundamental concept of OOP through Java programming
2. Write programs using console and dialog box.
3. Apply the concept of iterative, control, and array structure programming.
4. Construct classes, objects, methods and constructor.
5. Write programs in GUI environment

II. SOFTWARE/HARDWARE/EQUIPMENT NEEDED:

1. Eclipse
2. Computer unit

III. SAFETY GUIDELINES:

1. Make sure you have both an adjustable table and chair so that ergonomic accommodations can be made for each person using the computer.
2. The computer screen should be front and centre so neck turning is unnecessary.
3. Keep your lab space clean and organized.
4. Clean your lab bench and equipment, and lock the door before you leave the laboratory.
5. Never eat, drink, or smoke while working in the laboratory.

6. DO NOT TOUCH ANYTHING WITH WHICH YOU ARE NOT COMPLETELY FAMILIAR!!! It is always better to ask questions to laboratory technicians or to your instructors than to risk harm to yourself or damage to the equipment.

IV. THEORY

Computer program is read and execute by the machine sequentially or simply called sequential execution or sequential flow. However, codes can be inserted for execution depending on given conditions.

The repetition statements control a block of code to be executed for a fixed number of times or as long as a certain condition is met. In using repetitive statement the following terms must be observed:

counter - its a variable in which its function is to count the number of executions.

Initial value - is an assignment statement that is used to set the loop-control variable. This is executed only once.

Terminal value or condition - is a relational expression that determines when the loop will exit by testing the loop-control variable against some value.

Interval value - defines how the loop-control variable will change each time the loop is repeated.

The following are the different forms of repetitive statement.

1. The **for** statement

```
for (initial val ; terminal val ; interval)
{
        Statement:    //this will be executed if the condition is true
}
```

Example:

```
If(x=0 ; x<5 ; x++)
{
```

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

2. The ***while*** statement

```
Initial val;
while(terminal val)
{
        Statement:    //this will be executed if the condition is true
Interval;
}

Example:
X=0;
while(x<5)
{
        System.out.println(x);
X++;
}
```

3. The **do while** statement

```
do
{
        Statement:    //this will be executed before evaluating the
                      condition
}
while (condition)
        .
        .
        elseif (condition)
else
{
        Statement:    //this will be executed if the condition is false
}

Example:
x=0;
```

```
do
{
        System.out.println(x);
        x++;
}
while. (x<5);
```

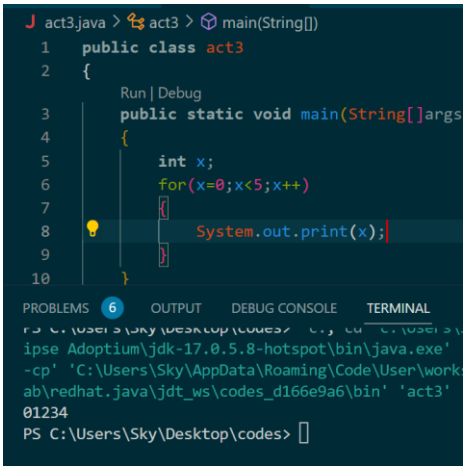
V. PROCEDURE:

1. Write the program below which is to display the value of the counter. Write comments after each line.

Program:

```
importjava.io.*;
importjava.util.*;

publicclass Loop2 {

      publicstaticvoid main(String[]args)throws Exception
      {
            int x;
            for(x=0; x<5;x++)
            {
                  System.out.println(x);
            }
      }
}
```

Output



2. Use the **while** loop statement to display for the given output

act3.java > act3

```
1   public class act3
2   {
        Run | Debug
3       public static void main(String[]args)
4       {
5           int x = 0;
6           while (x<5){
7               System.out.print(x+1 +". Marlon\n");
8               x++;
9           }
10      }
```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Sky\Desktop\codes>  c:; cd 'c:\Users\Sky\Desktop\co
ipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe' '-XX:+ShowCode
-cp' 'C:\Users\Sky\AppData\Roaming\Code\User\workspaceStorage\7
ab\redhat.java\jdt_ws\codes_d166e9a6\bin' 'act3'
1. Marlon
2. Marlon
3. Marlon
4. Marlon
5. Marlon
PS C:\Users\Sky\Desktop\codes>
```

Output:

1. Marlon
2. Marlon
3. Marlon
4. Marlon
5. Marlon

3. Use the**do while** loop statement to display for the same given output above.

act3.java > act3 > main(String[])

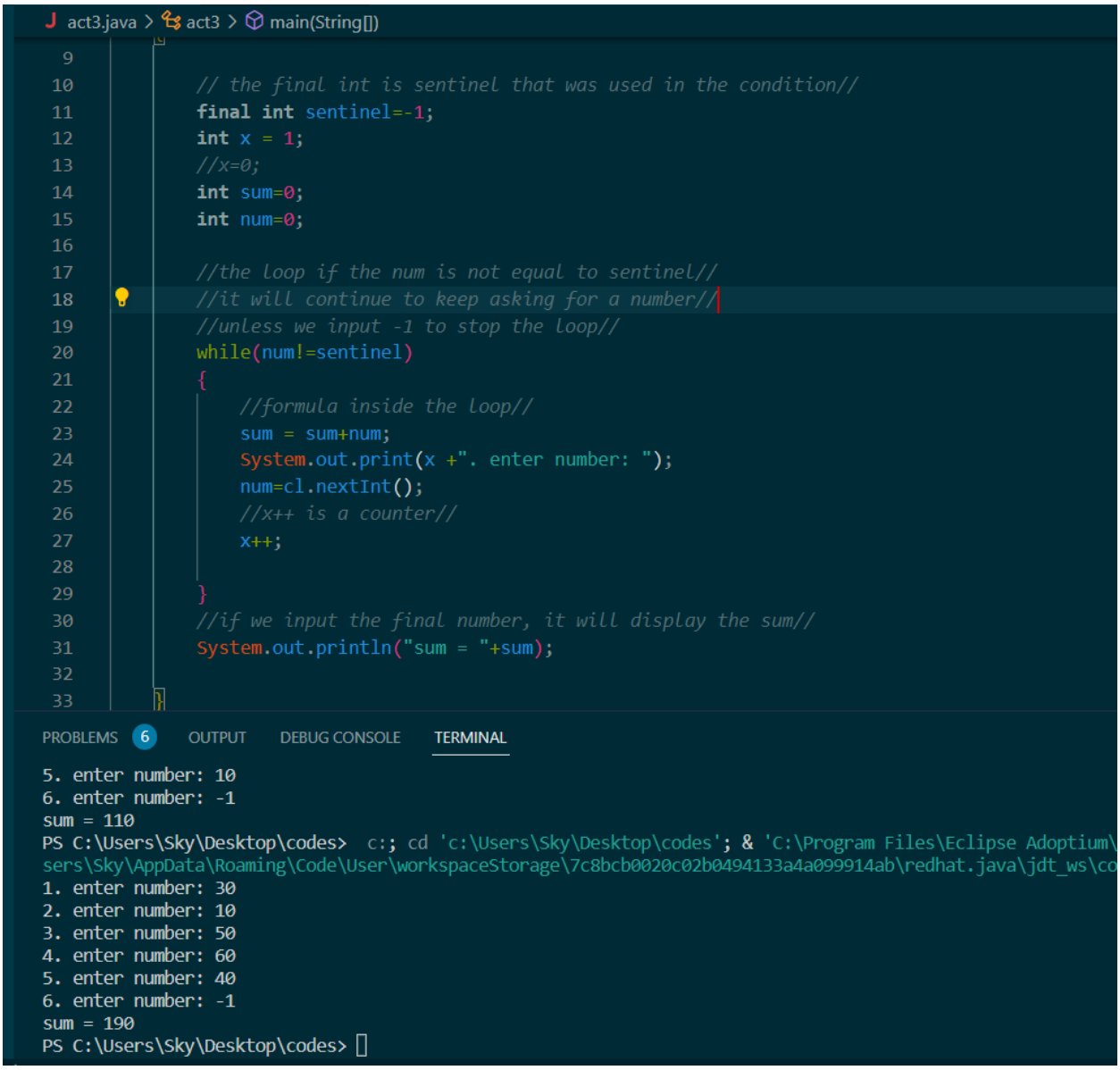
```
1   public class act3
2   {
        Run | Debug
3       public static void main(String[]args)
4       {
5           int x = 0;
6           do{
7               System.out.print(x+1 +". Marlon\n");
8               x++;
9           }
10          while (x<5);
11      }
```

PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Sky\Desktop\codes>  c:; cd 'c:\Users\Sky\Desktop\c
pse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe' '-XX:+ShowCode
p' 'C:\Users\Sky\AppData\Roaming\Code\User\workspaceStorage\7c
redhat.java\jdt_ws\codes_d166e9a6\bin' 'act3'
1. Marlon
2. Marlon
3. Marlon
4. Marlon
5. Marlon
PS C:\Users\Sky\Desktop\codes>
```

4. **Sentinel Loop:** write the program below and execute. Respond for the output by entering 5 different numbers and enter -1 as 6th input number. Explain the function of the inputted -1 in the program.

```
import java.io.*;
import java.util.*;

public class Loop1 {

        public static void main(String[]args)throws Exception
        {
                final int sentinel=-1;
                Scanner input =new Scanner(System.in);

                int x,num, sum;

                //x=0;
                sum=0;
                num=0;

                while(num!=sentinel)
                {
                        sum = sum+num;
                        //x++;
                        System.out.print("enter number: ");
                        num=input.nextInt();

                }
                System.out.println("sum = "+sum);

        }

}
```



VI. PROBLEMS/QUESTIONS:

1. Write a program that will implement the application of the **sentinel loop**.

2. Write a program to determine the number of PASSED and FAILED out of 15 inputted names with the corresponding grade. Restrict the inputted grade for each name in the range of 50 to 100, in which 75 is the passing grade. If the inputted grade is not on the

range, the program will remind the user of a message for the accepted range for the grade and allows the user to re-enter a grade.

3. Write a program to determine the number of MALE and FEMALE out of 15 inputted names with their corresponding gender. If MALE, the gender is 1, and if FEMALE, the gender is 0. Restrict the inputted gender for each name from 0 to 1. If the inputted gender is not on the range, the program will remind the user of a message for the accepted value for the gender and allows the user to re-enter the gender.

4. Write a program that prints all the factorial numbers up to an input number.

```
Enter any positive integer:
5
The factorial of 5 is 120
```

VII. INTERPRETATION/ANALYSIS OF DATA:

act3no1.java | act3no2.java | act3no3.java | act3no4.java

activity 3 > act3no1.java > act3no1 > main(String[])

```java
import java.util.*;


public class act3no1
{
    static Scanner cl = new Scanner(System.in);
    Run | Debug
    public static void main(String[]args)
    {
        clearScreen();

        // the string sentinel that was used in the condition//
        String sentinel= "yes";
        //counter//
        int x = 1;

        System.out.println();

        //the loop if the sentinel is not equal to no//
        //it will continue to keep counting//
        //unless we input "no" to stop the loop//
        while(sentinel.equalsIgnoreCase(anotherString: "no")!=true)
        {
            //count 1 to 4 every loop//
            System.out.print(x);x++;
            System.out.print("\n"+x);x++;
            System.out.print("\n"+x);x++;
            System.out.print("\n"+x);x++;
            //ask user to input yes or no//
            System.out.print(s: "\nwould you like to continue? (no to end): ");
            sentinel=cl.nextLine();



        }
        //if we input the final string, it will stop the counting//
        System.out.println(x: "\nCounting is stopped");

    }
```

Sentinel loop is having a loop with no end, user has full control of ending the loop.

Sentinel can be used as an integer or string

It tackles straight forward conditions if it's true or not.

act3no1.java | act3no2.java | act3no3.java | act3no4.java

activity 3 > act3no2.java > act3no2 > main(String[])

```java
public class act3no2 {
    static Scanner cl = new Scanner (System.in);
    Run | Debug
    public static void main(String args[]){
        clearScreen();

        //variables//
        int pass = 0, fail = 0;             //pass or fail counter//
        String[] name = new String[20];     //Array for name, can be used in the loop//
        int x;                              //counter//
        int[] y = new int[20];              //array y to add value to pass or fail//

        //first loop, limit to 15 input of name//
        for (x=1;x<=15;x++)
        {
            //ask for the user for names//
            System.out.print(x+ ". Input student's name: "); name[x]=cl.next();

            //nested loop//
            //ask for grades from the user//
            do{
                //input grade of student//
                System.out.print(s: "Input grade of student: "); y[x]=cl.nextInt();

                //if else statement, nested//

                if (y[x]<50 || y[x]>100)
                {
                    //if user input below 50 or above 100//
                    System.out.print(s: "Please input grades 50-100\n");
                }

                //counter for pass or fail//
                //if student failed, add count to fail//
                if(y[x]<75 && y[x]>=50)
                {
                    fail=fail+1;
                }
                //if student passed, add count to pass//
```

Variables are needed to be clear with using loop, especially multiple data input inside a loop

Array is the most suitable variable than can be used in loops, using counter variable to progress the data needed

Nested conditions are also quite tricky in this scenario, it can make the code more messy than it should.

Conditions for while loop should be straight forward and usually the opposite of the needed data to run smooth.

act3no1.java | act3no2.java | act3no3.java | act3no4.java

activity 3 > act3no3.java > act3no3 > main(String[])

```java
//variables//
//public global variables so it can be used in different void trees//
public static int gen1 = 0, gen2 = 0;
public static String[] name = new String[20];
public static int x;
public static int[] y = new int[20];

//main code//
Run | Debug
public static void main(String args[]){
    //clear the screen for much cleaner look//
    clearScreen();

    //first loop, limit to 15 input of name//
    for (x=1;x<=15;x++)
    {
        //ask for the user for names//
        System.out.print(x+ ". What's your name?: "); name[x]=cl.next();

        //nested loop//
        //ask for gender from the user//
        do{
            //input gender of name//
            System.out.print(s: "Input your gender (0 for MALE; 1 for FEMALE): ");
            y[x]=cl.nextInt();
            //separated code for cleaner look//
            ifs();

        }
        //limitation//
        //input gender will be 1 or 0//
        while(y[x]!= 1 && y[x]!=0);
        //every 2 rounds of loop, clear screen//
        if (x % 2 == 0){
            clearScreen();
        }
    }
```

Taking consideration of the previous problem of having a messy code, I made an alternative solution of making a sub-codes or trees where I can split the nested statements and condition in a different public static void.

Same with the previous code it function like a normal nested condition and loop codes.

It also has the function of telling the user to input correct data to progress with the code.

I also added a “clear screen” function where it reset the data on the screen to have a much cleaner look

activity 3 > act3no4.java > act3no4 > main(String[])

```java
import java.util.*;



public class act3no4 {
    static Scanner cl = new Scanner (System.in);

    //main code//
    Run | Debug
    public static void main(String args[]){

        //variables//
        int x,fact=1;
        int number;

        //ask user for input//
        System.out.print(s: "input a number: "); number=cl.nextInt();

        //loop for factorial//
        //formula using the counter variable//
        for(x=number;x>=1;x--){
            //print out the long version of vactorial//
            System.out.print("("+x+")");
            fact=fact*x;

        }
        //display the value of factorial//
        System.out.println("\nFactorial of "+number+" is: "+fact);
    }
}

```

Factorial use loop with counter and having the inputted data to be the condition to function correctly

Using the formula factorial=factorial*x; fact has an initial value of 1 since starting with 0 it will end with 0

X is the counter and it also start with 1, using the number variable to be the limit of the condition, I used the function of x-- to make it easier to visualize the factorial correctly, starting with the initial input to the number 1.

VIII. CONCLUSION/RECOMMENDATION:

The experiment made me more fluent with the language at hand, it's like relearning the previous languages that I've learned, it also practices my brain to analyze assignments and problem more fluently. The assignment also prompt my perspective on codes and how can I make it more aesthetically pleasing, not only for me but also to my future workmates. If it's easier for me to read, it is easier for them to understand. The idea of a better structure of codes is needed as much as how it can be run. Debugging is more easier because of that.

IX. REFERENCES:

Dale, Nell. (2018). Object-oriented data structures using Java.
Litvin, Maria. (2015). Java methods: : Object-oriented programming and data structures. 3rd Ed.
Baesens, Bart. (2015). Beginning java programming: : The object-oriented approach.
Gaddis, Tony. (2013).Starting out with Java: from control structures through objects. 5TH ed. Boston: Pearson