ITERATIVE CONTROL STRUCTURE

Loop

- One of a computer's most important attributes is the ability to perform repeatedly certain actions quickly, and loops are at the very heart of that.
- It is a repetition of a certain code segment/s in the program while the condition being evaluated remains to be true.
- It is used to iterate a part of the program repeatedly until specified condition is true.
- As soon as the condition becomes false, the loop automatically stops
- The while loop is considered a repeating "if-statement.

Kinds of Loop:

While Loop

- The DO/WHILE LOOP is a variant of the while loop.
- This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- The Do-While loop is an exit-controlled type of loop.
- Example:
  - ```
    int i = 0;
    while (i <5) {
            System.out.println(i);
            I++;
    }
    ```

Do-While Loop

- For a Do-While loop to work, the condition is not necessary to be met as this loop also works well for the first time even when the condition is not met.
- The compiler executor then enters the function execution block executes whatever is there within the block of statements, and then comes out to check the expression part where the condition is compared.
- If the condition is met, then the loop is reiterated; otherwise, the loop is exited from the block.
- The basic difference between the while and the do-while loop is that while the former one looks for the preconditions, the latter one targets the postconditions.
- Used when you want a code block to run repeatedly while the condition is met.
- A more compact format but more complicated version of a while loop.
- Example:
  - ```
    int i = 0;
    do {
            System.out.println(i);
            I++;
    } while (i < 5);
    ```

For Loop

- **Steps**
  - **Initializing Condition –** the variables to be used
  - **Testing Condition –** In the test condition, one of the counter variable variables is checked whether it is greater than or less than a certain quantity.
  - **Statement Execution –** in this phase, the print statement or the variable inside the for loop is executed making it easier to generate the output.
  - **Incrementing/Decrementing Condition –** in this phase, the loop control variable or the counter variable is incremented by 1 generally to move the code forward.
  - **Terminating the Loop –** when the condition doesn't satisfy in the testing condition phase, the loop closes and doesn't work anymore.
  - Example:
    - ```
      for (int i = 0; i < 5; i++) {
              System.out.println(i);
      }
      ```

JAVA ARRAYS

What is an Array?

- Stores multiple data items (elements) of the same data type
- Placed in a contiguous block of memory divided into slots
      ```

- Allows storage and manipulates data efficiently
- Elements are the individual values in an array.

Declaring Arrays

- **Declare with Values**
  datatype identifier[] = {val1, val2, val3, val4};
  String names[] = {"David", "Alenere", "Jasper"};

```
1
2 public class Arrays {
3
4     public static void main(String[] args) {
5         String studentName1[] = {"David",
6                                  "Goliath",
7                                  "Paul",
8                                  "Mark",
9                                  "Matthew"};
10
11        int numbers[] = {1,2,3,4,5,6,7,8,9,10};
12
13
14    }
15
16 }
17
```

- **Declare without Values**
  datatype identifier[] = new datatype[size];
  String names[] = new String[4];

```
1
2 public class Arrays {
3
4     public static void main(String[] args) {
5         /*String studentName1[] = {"David",
6                                  "Goliath",
7                                  "Paul",
8                                  "Mark",
9                                  "Matthew"};
10
11        int numbers[] = {1,2,3,4,5,6,7,8,9,10}; */
12
13        String employeeNames[] = new String[20];
14        int evenNumbers[] = new int [10];
15
16
17    }
18
19 }
20
```

- **Index** is a number that represents a position in a collection

VALUE	"David"	"Goliath"	"Paul"	"Mark"	"Matthew"
INDEX	0	1	2	3	4

Accessing Arrays

- Reading Array Elements
  Identifier[index];
  Names[0];

- Assigning Array Element
  Identifier[index] = value;
  Names[0] = "Miles";

Reading Array Elements

```
1
2 public class Arrays {
3
4     public static void main(String[] args) {
5         String studentName1[] = {"David",
6                                  "Goliath",
7                                  "Paul",
8                                  "Mark",
9                                  "Matthew"};
10        System.out.println(studentName1[2]);
11
12        int numbers[] = {1,2,3,4,5,6,7,8,9,10};
13        System.out.println(numbers[2] + numbers[5]);
14
15
16        String employeeNames[] = new String[20];
17        int evenNumbers[] = new int [10];
18
19
```

Assigning Array Element

```
public class Arrays {

    public static void main(String[] args) {
        String studentName1[] = {"David",
                                 "Goliath",
                                 "Paul",
                                 "Mark",
                                 "Matthew"};
        studentName1[0] = "Miles";
        System.out.println(studentName1[0]);
```

Assigning Array Element with User Input

```
1 import java.util.Scanner;
2 public class Arrays {
3     public static void main(String[] args) {
4
5         Scanner s = new Scanner(System.in);
6         String employeeNames[] = new String[20];
7         int evenNumbers[] = new int [10];
8
9         System.out.print("Enter Employee Name: ");
10        employeeNames[0] = s.nextLine();
11
12        System.out.println(employeeNames[0]);
13
14
```

```
<terminated> Arrays [Java Application] C:\Users\Acer\.p2\pool\plugins\org.
Enter Employee Name: Miles
Miles
```

2D JAVA ARRAYS

2D Array

- It's arrayed within an array.
- It basically looks like a table with rows and columns.
- Like 1D Arrays, it can only store one data type at a time.

Declaring 2D Arrays

- **Declare with Values**

```
datatype identifiers [][] = {
{val1, val2},
{val1, val2},
{val1, val2},
{val1, val2},
};
```

```
String users [][] = {
{"David", "david123"},
{"Alenere", "alenere123"},
{"Hezel", "hezel123"},
{"Jaymar", "jaymar123"},
};
```

```
package javaFundamentals;


public class TwoDimensionalArraysAndNestedForLoops {

    public static void main(String[] args) {

        String users[][] = {
                {"David","david123"},
                {"Alenere","alenere123"},
                {"Hezel","hezel123"},
                {"Jaymar","jaymar123"}
        };

    }

}
```

- **Declare without Values**

```
datatype identifiers [][] = new datatype[rows][cols];

String users[][] = new String[4][2];
```

PositiveNumb... | IterativeSt... | IterativeSt... | IterativeSt... | *Arrays.java

```
1  import java.util.Scanner;
2  public class Arrays {
3      public static void main(String[] args) {
4
5          int numbers[][] = new int [4][3];
6
7
8
9      }
10
11 }
12
13
```

Accessing 2D Array

```
String users[][] = {
        {"David", "david123"},
        {"Alenere", "alenere123"},
        {"Hezel", "hezel123"},
        {"Jaymar", "jaymar123"}
};
```

Index	0	1
0	"David" [0][0]	"david123" [0][1]
1	"Alenere" [1][0]	"alenere123" [1][1]
2	"Hezel" [2][0]	"hezel123" [2][1]
3	"Jaymar" [3][0]	"jaymar123" [3][1]

- **Read Values**
  System.out.println(identifier [row][col]);
  System.out.println(names[0][1]);

- **Write Values**
  Identifier[row][col] = value;
  Name[0][0] = "Ace";

```
1  package javafundamentals;
2
3
4  public class TwoDimensionalArraysAndNestedForLoops {
5
6      public static void main(String[] args) {
7
8          String users[][] = {
9                  {"David","david123"},
10                 {"Alenere","alenere123"},
11                 {"Hezel","hezel123"},
12                 {"Jaymar","jaymar123"}
13         };
14
15         int numbers[][] = new int[4][3];
16
17         System.out.println(users[1][1]);
18
19     }
20
21 }
```

Array Length

- In order to get the number of elements in an array, the length field of an array can be used.
- The length field returns the size of the array and is written as follows:
  arrayName.length

  String [] cars = {"Volvo", "BMW", "Ford", "Mazda"};
  System.out.println(cars.length);

```
1
2  public class Arrays {
3      public static void main(String[] args) {
4
5          String users[][] = {
6                  {"David","david123"},
7                  {"Alenere","alenere123"},
8                  {"Hezel","hezel123","mars"},
9                  {"Jaymar","jaymar123","jupiter","27"}
10         };
11
12         for (int row = 0;row < users.length;row++) {
13
14             for(int col = 0; col < users [row].length; col++) {
15                 System.out.println(row +" "+ col);
16
17             }
18             System.out.println();
19         }
20
21
22
23
24     }
25
26 }
```

JAVA METHODS

Methods

- Also known as subprograms or functions.
- Block of code that runs independently after being called.
- Program by itself – has its own input, output, and processes.
- Divides the entire program into smaller portions.

Types of Method

- **Static method**
  - Static is a keyword used to describe how objects are managed in memory.
  - It means that the static object belongs specifically to the class, instead of instances of that class.
  - **When should we use static methods?**
    - They are helpful for operations not dependent on any particular class instance, such as mathematical calculations or checking for equality.
  - **Static methods** can be convenient and help you write more concise code. However, they should be used sparingly and only when it makes sense.
  - In general, **static methods** are best used for utility classes and singletons. Otherwise, it is usually better to create an object of the class and call instance methods on that object.

Variable Scoping

- **Global Variables**
  - Variables declared within a class
  - It can be accessed within the whole class

```
1  package cc1;
2
3  public class Methods {
4
5      //Global Variables
6      static String section = "A";
7      static int num = 100;
8
9
10     public static void main(String[] args) {
11             sayNumber();
12             saySection();
13     }
14
15     static void saySection() {
16         System.out.println(section);
17     }
18     static void sayNumber() {
19         System.out.println(num);
20
21 }
22
23 }
24
```

- **Local Variables**
  - Variables declared inside a method, condition, loops, and any other block of code
  - It can only be accessible within that block of code

```
1  package cc1;
2
3  public class Methods {
4
5      public static void main(String[] args) {
6          greet();
7
8      }
9      static void greet(){
10         //LOCAL VARIABLE
11                 String greetings = "What's up yow!";
12                 System.out.println(greetings);
13     }
14
15 }
16
```

```java
package cc1;

public class Methods {

    public static void main(String[] args)
        String name = "Miles";
        if(name.equals("Miles")) {
            int age = 29;
            System.out.println(name);
            System.out.println(age);
        }

    }

}
```

- **Arguments/Parameters**
  - A value that needs to be passed on a Method so that the method can use that value and perform various operations on it.
- **Methods w/ Arguments**
  - modifiers returntype methodName(arguments) {
    //Do anything here
    }
  - static void say (String word) {
    System.out.println(word);
    }

```java
public class EasyMethod {

    public static void main(String[] args) {
        String x = "Miles";
        print("Hello, " + x);

    }
    static void print(String word) {
        System.out.println(word);
    }

}
```

  -

```java
public class EasyMethod {

    public static void main(String[] args) {
        add(5,10,10);

    }
    static void add(int num1, int num2, int num3) {
        System.out.println(num1 + num2 + num3);
    }

}
```

  -

```java
public class EasyMethod {

    public static void main(String[] args) {
        greet("Miles", 30);

    }
    static void greet(String name, int age) {
        System.out.println("Hello, " + name);
        System.out.println ("You are " + age + " Years Old!");
    }

}
```

- **Return method**
  - **Return Keyword**
    - Return keyword is used to return a value from the method.
    - It is used when a method has a result.
  - **Return Type**
    - The type of value that will be returned.
    - Return types are the same as the data types.
    - Int: return integers
      String: return strings
  - **Methods w/ Return**
    - Modifiers returntype methodName(arguments) {
      //do anything here
      Return value;

}

- static int add(int num1, int num2) {
  return num1 + num2;
  }

```
1 package cc1;
2
3 public class Methods {
4
5 public static void main(String[] args) {
6
7     int sum = add(5,2);
8     System.out.println(sum);
9
10 }
11     static int add(int num1, int num2) {
12         return num1+num2;
13     }
14
15 }
```

- **main() method**

```
import java.util.ArrayList;

public class WeatherAnalysisTool {

    public static void main(String[] args) {
        ArrayList<String> daysOfWeek = new ArrayList<>();
        ArrayList<Integer> temperatures = new ArrayList<>();

        // Populate the data
        String[] dayNames = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};
        int[] tempValues = {22, 22, 22, 22, 23, 23, 23};

        for (String day : dayNames) {
            daysOfWeek.add(day);
        }

        for (int temp : tempValues) {
            temperatures.add(temp);
        }

        // Convert array lists to 2D array
        String[][] weatherData = new String[2][daysOfWeek.size()];
        for (int i = 0; i < daysOfWeek.size(); i++) {
            weatherData[0][i] = daysOfWeek.get(i);
            weatherData[1][i] = String.valueOf(temperatures.get(i));
        }

        // Display the 2D array in a table form
        System.out.println("Day       | " + String.join(" | ", weatherData[0]));
        System.out.println("Temperature | " + String.join(" | ", weatherData[1]));
        System.out.println("-------------------------------------------");

        // Find day with lowest and highest temperature using for-loops
        String lowestDay = findDayWithLowestTemperature(weatherData);
        String highestDay = findDayWithHighestTemperature(weatherData);
        int lowestTemperature = Integer.parseInt(weatherData[1][findIndexForDay(weatherData, lowestDay)]);
        int highestTemperature = Integer.parseInt(weatherData[1][findIndexForDay(weatherData,
highestDay)]);

        System.out.println("Day with the lowest temperature: " + lowestDay + " with a temperature of " +
lowestTemperature + " °C");
        System.out.println("Day with the highest temperature is " + highestDay + " with a temperature of "
+ highestTemperature + " °C");
    }

    public static String findDayWithLowestTemperature(String[][] weatherData) {
        int minTemperature = Integer.MAX_VALUE;
        String minDay = "";

        for (int i = 0; i < weatherData[0].length; i++) {
            int temperature = Integer.parseInt(weatherData[1][i]);
            if (temperature < minTemperature) {
                minTemperature = temperature;
                minDay = weatherData[0][i];
            }
        }

        return minDay;
    }

    public static String findDayWithHighestTemperature(String[][] weatherData) {
        int maxTemperature = Integer.MIN_VALUE;
        String maxDay = "";

        for (int i = 0; i < weatherData[0].length; i++) {
            int temperature = Integer.parseInt(weatherData[1][i]);
            if (temperature > maxTemperature) {
                maxTemperature = temperature;
                maxDay = weatherData[0][i];
            }
        }

        return maxDay;
    }

    public static int findIndexForDay(String[][] weatherData, String day) {
        for (int i = 0; i < weatherData[0].length; i++) {
            if (weatherData[0][i].equals(day)) {
                return i;
            }
        }
        return -1;
    }
}
```

1. Iterative (Looping) Control Structures

One of a computer's most important attributes is the ability to perform repeatedly certain actions quickly, and loops are at the very heart of that. A loop is a repetition of a certain code segment/s in the program while the condition being evaluated remains to be true.

Loops usually have the following components:

1. ***Initialization*** of a variable or of several variables
2. ***Condition*** (that would evaluate to either true or false); the condition check is usually made on the current value of the variable initialized in (1) above.
3. ***Body of the loop*** which maybe a single statement or a group of statements
4. A ***change of state*** which is usually a statement inside the body of the loop that changes the contents of the variable(s)

Example: Printing the word "HELLO" fifty times.

Initialization:	counter = 1
Condition:	While counter <= 50
Body of the loop:	Print "HELLO"
Change of state:	counter = counter + 1

There are three available loop control structures in Java. These are:

1. `while` Loop
2. `for` Loop
3. `do-while` Loop

4.1. while Loop

The syntax of for the while loop is as follows:

```
<initialization>
while(<condition>)
{
      <statement1>;
          . . .
      <statementN>;
      <change of state>;
}
```

The body of the loop or statements under the while-loop will be repeatedly executed while the condition is `TRUE`. Otherwise, the loop will terminate and the next statement after the end of the while loop will be executed next.

Example: Considering the example given above, we have to write a program that will print the word "HELLO" fifty times.

```
//Printing HELLO 50 times

public class PrintHello{
     public static void main(String[] args){
          int counter;
          counter = 1;                    // Initialization
          while(counter <= 50){           // Condition

               System.out.println("HELLO"); // Body
               counter = counter + 1;  // Change of State
          }
     }
}
```

The program above can be modified easily to print the word "HELLO" any number of times. It is simply a matter of changing the value inside the condition.

Exercises: Evaluating the above program answer the following questions:

1. What will happen if we remove the initialization `counter = 1` in the program? This means that the value of **`counter`** is undefined or garbage.
2. What will happen if we remove the curly brackets enclosing the body of the loop?
3. What will happen if the programmer committed a typographical error, such that instead of pressing the less than symbol, the greater than symbol was pressed, i.e. the condition becomes `counter >= 50`?
4. What will happen if the programmer forgot changing the value of counter, i.e., `counter = counter + 1` was omitted.
5. Write a new program using a while loop where it will print the numbers 10 down to 1 in vertical way.

Note that the change of state need not always be an increment or decrement by 1. It can be of any value (also known as the step value) that is appropriate in the problem. Consider the next example:

Example: Program that will compute the sum of all numbers that are divisible by 5 from 0 to 100.

```
// by 5's program using while loop

public class ByFive{
     public static void main(String[] args){
          int num, sum;
          sum = 0;
          num = 0;

          while(num <= 100){
               sum = sum + num;
               num = num + 5;
          }
          System.out.println("sum = " + sum);
     }
}
```

Note that the change of state in the above program is an increment of 5. Once the value of `num` reaches 105, the body of the loop will terminate and the statement below it will be executed next, which is `System.out.println("sum = " sum);`.

Variable used for the change of state are not limited to integers only and need not always be incremental. Consider the following example:

```
// Step down using while loop
public class StepDown{
     public static void main(String[] args){
          double n = 10;
          while(n >= 1){
               System.out.println(n);
               n = n - 0.5;
          }
     }
}
```

Try simulating the output of the step down program written above.

4.2. for Loop

The syntax for a for-loop is as follows:

```
for([initialization]; [condition]; [change of state]){
     <statement1>;
          . . .
     <statementN>;
}
```

The for-loop is actually a "more compact" form of the while loop. The loop is executed as follows:

1. Perform the initialization
2. Check the condition
3. If the condition results to a `TRUE` value, the statement/s inside the for-loop (body) will be executed. Else proceed to #6
4. Change of state
5. Goes back to #2.
6. Exit the loop and the statement after the end of the for-loop will be executed.

Example: Print the word "`HELLO`" fifty times using for-loop:

```
//HELLO 50 times using for-loop example

public class PrintHello{
     public static void main(String[] args){
          int counter;

          for(counter = 1; counter <= 50; counter++){
               System.out.println("HELLO");  // Body
          }
     }
}
```

Note that the initialization, condition, and the change of state are only found in one line. The body of the for-loop like the while-loop is enclosed in a pair of curly braces. If there is only one statement representing the body of the loop, then the pair of curly brackets can be optional.

`counter++` is also the same as `counter = counter + 1`

Example: Print all EVEN numbers from 100 down to 0

```
// Even from 100 down to 0 using for-loop

public class EvenNumbers{
     public static void main(String[] args){
          int i;

          for(i = 100; i >= 0; i = i - 2)
               System.out.println(i);
     }
}
```

Example: A program that display the Fibonacci series.

```
// Fibonacci series using for-loops
import java.io.*;
public class FibonacciNumbers{
     public static void main(String[] args){
       int prev = 0, next = 1, ans = 0;
       int i, num = 0;
       String input = " ";

       BufferedReader in = new BufferedReader(new
                           InputStreamReader(System.in));

       System.out.print("Input a number: ");
       try{
          input = in.readLine();
       }catch(IOException e){
          System.out.println("Error!");
       }
       num = Integer.parseInt(input);
       System.out.println("Fibonacci series is: " + num);

       for(i = num; i >=0; i--){
          System.out.print(ans);
          prev = next;
          next = ans;
          ans = prev + next;
       }
     }
}
```

Sample Output: The Fibonacci of 10

```
Input a number: 10
The Fibonacci series of 10 is:
0 1 1 2 3 5 8 13 21 34 55 _
```

4.3. do-while Loop

The syntax for a Do-While-loop is as follows:

```
<initialization>;

do{
    <statement1>;
        . . .
    <statementN>;
    <change of state>;
}while(<condition>);
```

The Do-While is executed as follows:

1. Performs the initialization.
2. Performs the body of the loop
3. Change of state
4. Checks the condition. If the condition results to a TRUE value, the statement/s inside the do-while Loop (body) will be executed until condition will be FALSE.
5. If condition is FALSE, exit the loop and the statement after the end of the do-while Loop will be executed.

Example: Print the word “HELLO” fifty times using do-while-Loop:

```
//HELLO x 50 using do-while-loop example

public class PrintHello{
     public static void main(String[] args){
          int counter;
          counter = 1;              // Initialization
          do{
            System.out.println(“HELLO”);      // Body
            counter = counter + 1;    // Change of State
          }while(counter <= 50);         // Condition
     }
}
```

Note that in a `do-while` Loop, the body is executed at least once before the condition is checked. Unlike in the `while` and `for` loops, the condition is first checked before the body of the loop can be executed.

Example: Print all EVEN numbers from 100 down to 0

```
// Even from 100 down to 0 using do-while-loop

public class EvenNumbers{
     public static void main(String[] args){
          int i;
          i = 100;
          do{
               System.out.println(i);
               i = i - 2;
          }while(i >= 0);
     }
}
```

4.4. Other implementation of loops

Loops are not confined on processing numeric data, the program will depend on the problem that is being solved. Let us consider the problem below.

```
// Interactive program using loops
import java.io.*;
public class InteractiveProgram{
  public static void main(String[] args){
      int a, b = 0, sum;
      char again;
      String input = " ";
      BufferedReader in = new BufferedReader(new
                              InputStreamReader(System.in));
      do{
        sum = 0;
        for(a = 0; a < 10; a++){
          System.out.println("Input number " + a + " of 9: ");
          try{
           input = in.readLine();
          }catch(IOException e){
           System.out.println("Error!");
          }
          b = Integer.parseInt(input);
          sum = sum + b;
        }
        System.out.println("The sum = " + sum);
        System.out.println("Do you want to try again? (Y/N)");
        try{
           input = in.readLine();
         }catch(IOException e){}
         again = input.charAt(0);
      }while(again == 'Y' || again == 'y');
      System.out.println("Have a nice day!");
  }
}
```

4.5. Counters

A counter is a variable that is used to keep track of the count (frequency) of a certain group of items. Usually,

- Its data is integer
- It is initialized to a value of 0
- Incremented by 1 inside a loop

Example: Write a program that will ask the user to input 10 integers. The program should output how many of the data are positive. Assume that zero is a positive integer.

```
// Counter program
import java.io.*;
public class CounterProgram
  public static void main(String[] args){
      int i=0, num, ctr_positive;
      String input = " ";
      ctr_positive = 0;

      BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));

      for(i = 0; i < 10; i++){
        System.out.println("Input number " + i + " of 9:", i);
        try{
            input = in.readLine();
        }catch(IOException e){
            System.out.println("Error !");
        }
        num = Integer.parseInt(input);
        if(num >= 0){
            ctr_positive = ctr_positive + 1;
        }
      }
      System.out.println("Positive integers= " + ctr_positive);
  }
}
```

Exercise:

1. Modify the previous program such that it will also count and print the number of negative integers that were inputted by the user.
2. Create a new program that will ask the user to input **n** integer values then print the number of ODD and EVEN numbers.
3. Create a program the will display all numbers that are divisible by 5 and count how many of them are there in the range of a signed integer data type.

4.6. Accumulators

An **accumulator** is a variable that is used to keep track of the accumulated value of a certain group of items. An accumulator

- May have a data type of `int`, `float` or `double`
- It is usually initialized to a value of 0
- Changes by assuming the sum of the current value of the accumulator and the value of another variable

Example: Write a program that will ask the user to input **n** integers. The program should output the sum of all the input data.

```
// Accumulator Sample
import java.io.*;
public class AccumulatorSample{
  public static void main(String[] args){
      int n, i, num, sum;
      String input = " ";
      sum = 0;

      BufferedReader in = new BufferedReader(new
                              InputStreamReader(System.in));
      System.out.println("How many numbers to process? ");
      try{
        input = in.readLine();
      }catch(IOException           e){
        System.out.println("Error!");
      }
      n = Integer.parseInt(input);
      for(i = 1; i <= n; i++)
      {
        System.out.println("Input number " + i + " of " + n);
        try{
          input = in.readLine();
        }catch(IOException e){
          System.out.println("Error!");
        }
        num = Integer.parseInt(input);
        sum = sum + num;
      }
      System.out.println("Sum of all input data is = " + sum);
  }
}
```

Exercises:

1. Modify the previous program such that it will compute and output the sum of positive numbers and the sum of negative numbers.

2. Create a new program that will ask the user to input how many students are there in a class. Thereafter, the user will be asked to input the final grade for each student. The program should determine the average of all the grades of the student in that class. How many of them passed, and how many of them failed.

   Assume that the passing grades are from 75 to 99, and failing grades are from 70 to 74.

UNIT 3

Arrays and Introduction to Methods

1. Java Arrays

An array stores multiple data items of the same type, in a contiguous block of memory, divided into a number of slots. Furthermore, **array** is a capability of programming languages wherein one variable can store a list of data and manipulate these data more efficiently.

In simple words:

- An array is a group of elements with the same data type (homogeneous).
- An array is characterized by its name, dimension, size, and element data type.
- An array has a dimension:
  - If dimension is 1, we say that the array is a one-dimensional; Also known as a list.
  - If dimension is 2, we say that the array is a two-dimensional; Also known as a table.
  - Array having more than one dimension is also known as multi-dimensional array.

In our course, we will just discuss one and two dimensional arrays. But in the Java language, you can have more than two dimensions for an array.

1.1. Declaring a One-Dimensional Array

The syntax for a one-dimensional array:

```
<data_type> <array_name>[];
          or
<data_type>[] <array_name>;
```

After declaring, you must create the array and specify its length using a **constructor.** The process of using a **constructor** to create an array is called **instantiation**.

Sample Declaration:

```
//declaration
int number[];
//instantiation
number = new int[10];
```

It can also be written as:

```
//declare and instantiate
int number[] = new int[10];
```

The declaration tells the Java Compiler that the identifier `number` will be used as the name of the array containing integers and creates or instantiate a new array containing 10 elements.

Examples:

```
char name[] = new char[35];
int grades[] = new int[50];
float area[] = new float[5];
double data[] = new double[10];
```

Note: array elements not explicitly initialized will have garbage values.

To reference an element in a one-dimensional array:

```
<array_name>[<index>]
```

Key Points when referencing elements in an array in Java:

- The range of values that you can use as array index is from 0 to <size-1>
- The first element is always at index 0
- The last element is always at index <size -1>
- Index cannot be negative
- Index cannot be a float or double (real) number, only integer values

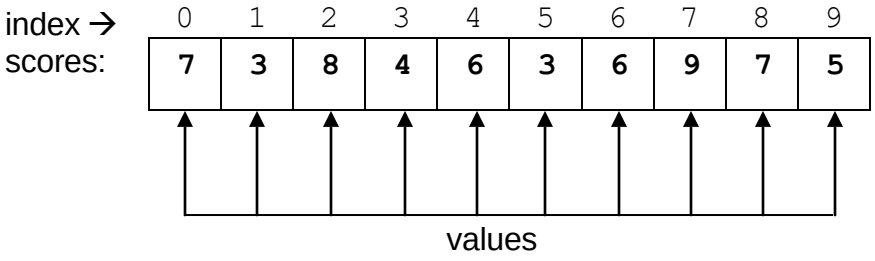
Example: A list of 10 scores in a quiz:

```
int scores[] = new int[10];
```

- The range of index values is from 0 to 9
- The first element is at `scores[0]`
- The last element is at `scores[9]`

```
scores[0] = 7
scores[1] = 3
scores[2] = 8
scores[3] = 4
scores[4] = 6
scores[5] = 3
scores[6] = 6
scores[7] = 9
scores[8] = 7
scores[9] = 5
```

Graphical representation of scores array having 10 as its size:



You can also initialize values of an array during declaration. Examples are:

```
/*creates an array of boolean variable with identifier
values initialized to True and False */

boolean values[] = {true, false};

/*creates an array of 4 double values (100, 90, 80,
75) with identifier name grades*/

double[] grades = {100, 90, 80, 75};

//creates an array of Strings with identifier days

String days[] = {"Mon", "Tues", "Wed", "Thurs", "Fri",
                 "Sat", "Sun"};
```

Example: Create a program that will declare a one-dimensional array with a size of 10. Initialize the array and let the user input 10 quiz scores as given above, thereafter compute the average of these 10 scores. Assume that the perfect score per quiz is 10 points.

```
// 10 quiz scores using 1-D array
import java.io.*;
public class TenQuizScores{
  public static void main(String[] args){
   int scores[] = new int[10];
   int i, sum = 0, num = 0;
   double ave = 0;
   String input = " ";

   BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));

   // Initialize list to zero
   for(i = 0; i < 10; i++){
     scores[i] = 0;
   }
   // Input 10 scores
   for(i = 0; i < 10; i++){
     System.out.print("Input score " + i + " = ");
     try{
       input = in.readLine();
     }catch(IOException e){
       System.out.println("Error!");
     }
     num = Integer.parseInt(input);
     scores[i] = num;
   }
   // Print inputted scores
   for(i = 0; i < 10; i++){
     System.out.println("score " + i + " = "+ scores[i]);
   }
   // Compute sum of all 10 scores
   for(i = 0; i < 10; i++){
     sum = sum + scores[i];
   }
   // Compute the average
   ave = (double)sum / 10; // type casts (double)sum

   System.out.println("Sum of all scores is = " + sum
                         + " over 100");
   System.out.println("The average is = " + ave);
  }
}
```

Note: `ave = (double)sum/10`. If we remove the cast `(double)` before the variable `sum`, the result will be an integer division, since `sum` is an integer and `10` is also an integer. The fraction part will be dropped. In order to retain the fraction part for the `ave` variable, we need to preface the variable with a new type in parenthesis. This is called type casting in Java.

1.2. Declaring Two-Dimensional Array

The syntax in declaring a two-dimensional array is as follows:

```
<data_type> <array_name>[<rowsize>][<colsize>];
              Or
<data_type>[<rowsize>][<colsize>] <array_name>;
```

The **instantiation** of a two-dimensional array is the same as one-dimensional array except that the size would contain two value, size for the row and size for the column.

Sample Declaration:

```
//declaration
int number[][];
//instantiation
number = new int[5][10];
```

It can also be written as:

```
//declare and instantiate
     int number[][] = new int[5][10];
```

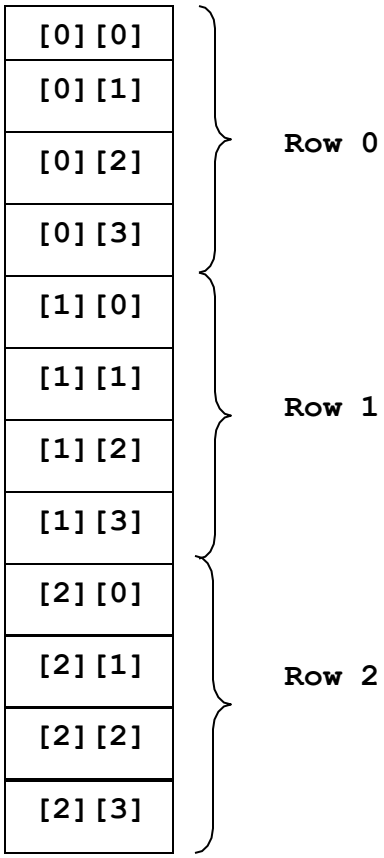
Example:

```
int table[][] = new int[3][4];
```

Graphical representation of `table[][]` of `int[3][4]`:

	Col 0	Col 1	Col 2	Col 3
Row 0	**[0][0]**	**[0][1]**	**[0][2]**	**[0][3]**
Row 1	**[1][0]**	**[1][1]**	**[1][2]**	**[1][3]**
Row 2	**[2][0]**	**[2][1]**	**[2][2]**	**[2][3]**

Graphical representation of `table[][]` of `int[3][4]` using row major implementation:



To reference an element in a two-dimensional array:

<array_name>[<row_index>][<column_index>]

Key Points when referencing elements in a two-dimensional array in Java:

- The range of row index is from 0 to <rowsize> - 1
- The range of the column index is from 0 to <columnsize> - 1
- The first element is always at index <array_name>[0][0]
- The last element is at index <array_name>[rowsize-1][colsize-1]
- Index cannot be negative
- Index cannot be a float or double (real) number, only integer values
- Normally, a double loop is used as control structure for processing two-dimensional array.
- Normally, the processing is done row-by-row, and column-by-column within the same row.

Example: A table 3 x 4 of integer data :

```
int table[][] = new int[3][4];
table[0][0] = 2;
table[0][1] = 4;
table[0][2] = 5;
table[0][3] = 7;
table[1][0] = 1;
table[1][1] = 6;
table[1][2] = 9;
table[1][3] = 3;
table[2][0] = 4;
table[2][1] = 6;
table[2][2] = 2;
table[2][3] = 8;
```

Graphical representation of `table[][]` of `int[3][4]`:

	Col 0	**Col 1**	**Col 2**	**Col 3**
Row 0	**2**	**4**	**5**	**7**
Row 1	**1**	**6**	**9**	**3**
Row 2	**4**	**6**	**2**	**8**

```
//Table 3 x 4 program
public class Table3x4{
     public static void main(String[] args){
       int table[][] = new int[3][4];
       int i, j;
       table[0][0] = 2;
       table[0][1] = 4;
       table[0][2] = 5;
       table[0][3] = 7;
       table[1][0] = 1;
       table[1][1] = 6;
       table[1][2] = 9;
       table[1][3] = 3;
       table[2][0] = 4;
       table[2][1] = 6;
       table[2][2] = 2;
       table[2][3] = 8;
       // print table in matrix format
       for(i = 0; i < 3; i++){
          for(j = 0; j < 4; j++){
            System.out.print(" " + table[i][j]);
          }
          System.out.println();
       }
     }
}
```

Using the same table, create a program that will initialize the table to zero and ask the user to input values. Thereafter, the program will print the inputted values in matrix form. The program should also display the sum of all the values inputted.

```
//Table 3 x 4 modified program
import java.io.*;
public class Table3x4Modified{
  public static void main(String[] args){
     int table[][] = new int[3][4];
     int i, j, sum = 0, num;
     String input = " ";

     BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));
     // Initialize table to zero
     for(i = 0; i < 3; i++){
          for(j = 0; j < 4; j++){
               table[i][j] = 0;
          }
     }
     // Input values
     for(i = 0; i < 3; i++){
       for(j = 0; j < 4; j++) {
         System.out.print("Input table [" + i + "]["+
                               j + "] = ");
         try{
           input = in.readLine();
         }catch(IOException e){
           System.out.println("Error!");
         }
         num = Integer.parseInt(input);
         table[i][j] = num;
       }
     }
     // computing the sum
     for(i = 0; i < 3; i++){
          for(j = 0; j < 4; j++){
               sum = sum + table[i][j]);
          }
     }
     // print table in matrix format
     for(i = 0; i < 3; i++){
       for(j = 0; j < 4; j++){
          System.out.print(" " + table[i][j]);
       }
       System.out.println();
     }
     System.out.println("Sum = " + sum);
  }
}
```

Exercises:

Write a program that will create a 2-dimensional array with a row size = 10 and a column size = 10 also. Or simply a 10 x 10 table, thereafter the program should:

1. Initialize the array to zero.
2. Ask the user to input integer values.
3. Display the table in matrix form.
4. Display the sum of all elements.
5. Display the average of all elements.
6. Display the number of negative elements in the table.
7. Print only the elements on the main diagonals of the table.
8. Modify the program and display the sum of the elements on a specified row.
9. Modify the program and display the sum of the elements on a specified column.
10. Assume three matrices of the same size, say matrix A, B and C. Write a program that will add the two matrices A and B and store the sum to C. Matrix addition is done component-wise, i.e., `C[i][j] = A[i][j] + B[i][j]` where `i,j` are the row and column index respectively.

1.3 Array Lengths

In order to get the number of elements in an array, the `length` field of an array can be used. The `length` field returns the size of the array and is written as follows:

```
arrayName.length
```

Sample Program: Using length field

```
public class ArrayLength{
     public static void main(String[] args){
          int[] scores = new int[50];
          int i;

          for(i = 0; i < scores.length; i++)
               System.out.print(scores);
     }
}
```

Note: It is better to use the `length` field as a condition statement of a loop to allow the loop to adjust automatically to different - sized arrays.

2. Java Methods

To know more of what a method is, let us first consider the following scenario. Let us assume that you were a programmer and you were asked to write a program that will be used to handle Automated Teller Machine transactions. Such transactions includes: (1) balance inquiry, (2) deposit and (3) withdraw. Aside from these, the user should be presented a menu showing there transaction as possible options. For a non-trivial program like these, it is advise to write the codes under one method which the **`main()`** method.

It is better to subdivide the programs into smaller portions – which we call methods. Each method should be able to solve part of the bigger problem. In the scenario above, one method will handle the generation of the menu, another method will be in-charge of handling the balance inquiry, another method will be used for deposit transaction, and a method that will handle withdrawal transactions. Notice that each method does a specified job of other methods.

In a more concrete term, a method is basically a program by itself – it has inputs, output, and will perform some kind of processing steps.

In computer programming, methods are also called subprograms. An aspiring computer programmer should be able to determine based from the problem statements, what methods are needed and how to relate these methods in order to solve the problem.

Characteristics of Methods

- It can return one or no values
- It may accept as many parameters it needs or no parameter at all
  - **Parameters** are also called function arguments
- After the method has finished execution, it goes back to the method that called it

2.1 Declaring Static Methods

Static methods are methods that can be invoked without instantiating a class. These methods belong to the class as a whole and not to a certain instance of a class. Static method is defined by the keyword `static` and can only be invoked inside another static method.
To declare methods, we follow the syntax:

```
<access_modifier> static <return_type>
                  <method name>([parameters]){
            method body
}
```

Where,

- **access_modifier** can either be public, private, protected (if no access is present, the default modifier is in effect).
- **static** is a keyword used to indicate that the method is a static method
- **return_type** can be the different data types.
- **method_name** is any name given by the programmer to identify the method.
- **parameters** are arguments that is passed from one-method to another (optional if no arguments are to be passed).

Sample Method Declarations:

```
public static int withParameter(int  n1, int n2){
     //body of the method
}

public static void withOutParameters(){
     //body of method
}
```

2.2 Calling Static Methods

To call a static method, the syntax to follow is,

```
ClassName.staticMethodName(params);
          Or
staticMethodName(params);
```

Sample Program: Program that print the text “Hello World!” where “Hello” is to be printed by one method and “World!” is to be printed by another method.

```
//Declaring and Calling Methods
public class UsingMethods{
     public static void printHello(){
          System.out.print(“Hello “);
     }
     public static void printWorld(){
          System.out.println(“World!”);
     }
     public static void main(String[] args){
          printHello();//same as UsingMethods.printHello
          printWorld();//same as UsingMethods.printWorld
     }
}
```

The previous program is interpreted as,

- `//Declaring and Calling Methods`
  - A program comment
- `public class UsingMethods{`
  - The name of the class (and the file)
- `public static void printHello(){`
  - A user – defined static method without any return value whose method name is printHello without any parameters
- `System.out.print("Hello ");`
  - Prints the text Hello on screen
- `}`
  - Ends the method body
- `public static void printWorld(){`
  - A user – defined static method without any return value whose method name is `printWorld` without any parameters
- `System.out.println("World!");`
  - Prints the text World! on screen
- `}`
  - Ends the method body
- `public static void main(String[] args){`
  - the main method (pre-defined method) with 1 parameter
- `printHello();`
  - Calling the method `printHello` and performing statements found therein
- `printWorld();`
  - Calling the method `printWorld` and performing statement found therein
- `}`
  - Ends the main method body
- `}`
  - Ends the class

Example: Calling **`hello()`** function three times inside **`main()`**

```
// calling hello() three times in main()
public class Hello3x{
     public static void hello(){
          System.out.println("Hello World!");
     }
     public static void main(String[] args){
          hello(); // 1st method call
          hello(); // 2nd method call
          hello(); // 3rd method call
     }
}
```

Multiple invocation of the same method must be placed inside a loop.

```
// calling hello() 1000 times in main()

public class Hello1000x{
     public stati void hello(){
          System.out.println("Hello World!");
     }
     public static void main(String[] args){
          int i;

          for(i = 0; i < 1000; i++){
               hello();
          }
     }
}
```

Exercise: What will happen if we write the **`hello()`** function after the **`main()`** function? Will the program behave as what is expected? If not, how will you remedy the situation where functions are written after the **`main()`** method?

2.2.1. Calling a Method Inside another Method

A method can be invoked inside another method aside from `main()`.

Example:

```
// function call beside main()
public class FunctionCallBesideMain{
     public static void printHello(){
          System.out.print("Hello");
     }
     public static void printSpace(){
          System.out.print(" ");
     }
     public static void printWorld(){
          System.out.println("world!");
     }
     public static void allTogetherNow(){
          printHello();
          printSpace();
          printWorld();
     }
     public static void main(String[] args){
          allTogetherNow();
     }
}
```

2.3. Passing Variables in Methods

There are two ways of passing data to methods, one is to pass-by-value and the other is to pass-by-reference.

2.3.1. Pass-by-value

When pass-by-value occurs, the method makes a copy of the value of the variable passed to the method.

Example: Passing a value to a method

```
//Pass by value
public class PassByValue{
      //Method with parameter
      public static void methodWithParam(int j){
            //print value of j
            System.out.println("Value of j = " + j);
      }
      public static void main(String[] args){
            int i;
            i = 50;

            //call methodWithParam passing value of i
            methodWithParam(i);
            //print value of i
            System.out.println(i);

      }
}
```

Pass i as a parameter which is copied to j

Return to the caller and perform the statement after the method call

The output of the above program would be:

```
Value of j = 50;
Value of i = 50;
```

Supposed that you add the `statement j = 100;` before the `System.out.println("Value of j = " + j);` inside `methodWithParam(int j)`, the output would be:

```
Value of j = 100;
Value of i = 50;
```

Note: The method cannot modify the original value of the variable being passed. By default, all primitive data types when passed to a method are pass-by-value.

Sample program passing two parameters.

```
//Pass by value
public class PassByValue{
     //Method with parameter
     public static void methodWithParam(int j, int k){
          //print value of j
          System.out.println("Value of j = " + j);
          System.out.println("Value ok k = " + k);
     }
     public static void main(String[] args){
          int i;
          i = 50;

          /*call methodWithParam passing value of i
            copied to both parameters of
            methodWithParam*/
          methodWithParam(i, i);
          //print value of i
          System.out.println(i);
     }
}
```

The output of the above program would be

```
Value of j = 50
Value of k = 50
Value of i = 50
```

2.3.2. Pass-by-reference

When pass-by-reference occurs, the reference to an object is passed to the calling method. But unlike pass-by-value, the method can modify the actual object that the reference is pointing to since the location of the data pointed to is the same.

Example: Passing reference to a method

```
public class PassByReference{
     public static void methodWithParam(int[] arr){
          //change values of array
          for(int i = 0; i<arr.length; i++)
               arr[i] = i + 50;
     }

     public static void main(String[] args){
          //create array of integers
          int numbers[] = {10, 20, 30};

          //print array values
          for(int i = 0; i < numbers.length; i++)
               System.out.println(numbers[i]);

          /*call methodWithParam ans  pass reference to
            array */
          methodWithParam(numbers);

          //print array values again
          for(int i = 0; i < numbers.length; i++)
               System.out.println(numbers[i]);
     }
}
```

Pass numbers as parameter which whose memory reference is pointed to by variable arr

Return to the caller and perform the statement after the method call

The output of the above program is:

```
10
20
30
60
70
80
```

Note: Instead of copying the values passed by the method, the pointer to the memory reference is the one being passes since it does not involve a primitive data type.

2.4. Methods with return types

So far, example given regarding methods makes use of **`void`** as a return type. The type **`void`** should be used as the return type only when the method does not return anything. However, there are cases

wherein the method will have to return a character, an integer, a float, or a double data type depending on the problem being solved.

Example: A program that will return integer type.

```
// method with return type int
public class MethodWithReturnType{

  //declaring method with int as return type
  public static int computeSum(int x, int y){
     int z;
     z = x + y;
     return z;
  }

  public static void main(String[] args){
     int a, b, c;

     /*call method computeSum passing 8 and 13 as
       parameters*/
     System.out.println("Sum = " + computeSum(8, 13));

     /*calling method computeSum passing 100 and 200
       as parameters and storing the result to variable
       c*/
     c = computeSum(100, 200);
     System.out.println("Sum = " + c);

     a = 22;
     b = 14;
     /*call method computeSum passing values of variables
       A and b as parameters*/
     System.out.println("Sum = " + computeSum(a, b));

  }
}
```

The output of the above program would be:

```
Sum = 21
Sum = 300
Sum = 36
```

Exercise:

1. What will happen if the statement **`return z`** was omitted?
2. Write a method named **`Difference()`** that accepts two integer parameters named **`x`** and **`y`**. The method should compute and return the value of the difference of **`x`** and **`y`**. Call this method inside **`main()`**.
3. Write a method named **`Product()`** that accepts two floating point parameters named **`x`** and **`y`**. The method should compute and return the value of the product for **`x`** and **`y`**. Use **`double`** as the return type. Call this method inside **`main()`**.

Example: Write a method that will accept an integer variable as parameters, thereafter, the method should return 1 if the integer is positive, otherwise it should return 0.

```
// function to return 1 if positive otherwise return 0

public static int numberPositive(int n){
     int result;

     if (n > = 0)
          result = 1;

     else
          result = 0;

     return result;
}
```

Another way to implement this method is as follows:

```
// Another implementation

public static int numberPositive(int n)
{
     if (n >= 0)
          return 1;
     else
          return 0;
}
```

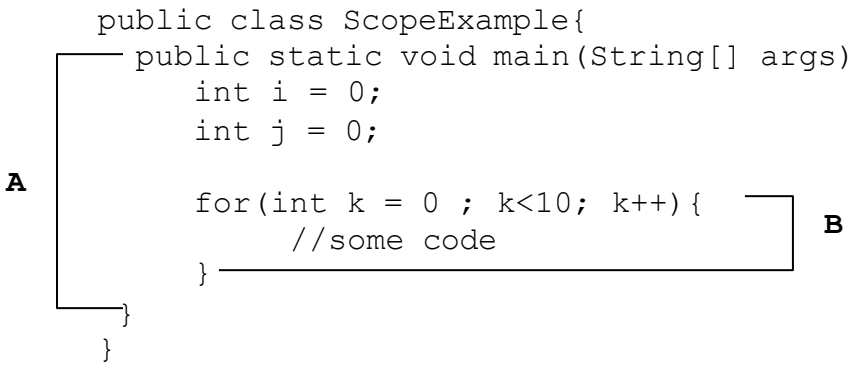
2.5 Important things to remember about methods

1. If the method will not return anything, use **`void`** as the return type.
2. If the method will return a value, the appropriate data type should be specified.
3. If the method will return a value, don't forget to use the return statement inside the method.
4. The data type of the value that appears after the return statement must be compatible with the return data type.
5. You can use any name for the method as long as you follow the naming convention and do not use Java keywords as method names.
6. A method may have zero, one or more parameters.
7. If the method does not have any parameter, leave the area in between the parentheses blank.
8. If the method has parameters, specify their data types and names.
9. Parameters should be separated by comma.
10. The name of the parameters does not matter. Always remember that the n*umber of parameters, their respective data types, and their sequence* are the things that actually matter!
11. When calling a method, constants, expressions and variables maybe used as actual parameters.
12. A method can return at the most only one value at any given point in time.

3. Scope of Variable

The **scope** determines where in the program the variable is accessible. The scope also determines the lifetime or how long the variable can exist in memory. The placement of the variable, or where the variable is declared, determines the scope for that variable.

For example:



The sample code snippet written above represents two scopes indicated by the lines and letter representing the scope.

Scope A – the scope of variables `i` and `j` is true to the whole `main()`, meaning, both variables `i` and `j` can be used inside the `main()` block

Scope B - variable `k` can only be used inside the `for loop` block.

Given a complete sample program with method,

```
public class PassByReference{
  public static void main(String[] args){
     //create an array of integers
     int numbers[] = {10, 20, 30};

     //print array values
     for(int i = 0; i < number.length; i++){
     System.out.println(numbers[i]);
     }

     //call method and pass reference to array
     methodWithParam(numbers);

     //print array values again
     for(int i = 0; i < numbers.length; i++){
       System.out.println(numbers[i]);
     }
  }

   public static methodWithParam(int[] arr){
     //change value of array
     for(int i = 0; i< arr.length; i++){
       arr[i] = i + 50;
     }
   }
}
```

B

A

C

D

E

Inside the `main()` method:

`numbers[]` – scope A
`i` in B – scope B
`i` in C – scope C

Inside the `methodWithParam()` method:

`arr[]` – scope D
`i` in E – scope E

REFERENCES

A. Books

Cadenhead, R., Lemay, L. (2007), **Sams Teach yourself Java 6 in 21 Days, 5th Edition**, Sams Publishing.

Deitel, P.J., Deitel, H.M. (2007), **Java: How to Program, 7th Edition**, Prentice Hall.

Bates, B., Sierra, K. (2005), **Head First Java, 2nd Edition,** O' Reilly.

Schildt, H. (2005), **Java: The Complete Reference, 5th Edition** , McGraw-Hill.

Williams, A. (2002), **Java 2, Core Language, Little Black Book**, Paraglyph.

B. Websites

The Java Tutorial: A Practical Guide for Programmers.
Available at http://java.sun.com/docs/books/tutorial

Java in a Nutshell. Chapter 4: The Java Platform.
Available at http://www.unix.org.ua/orelly/java-ent/jnut/ch04_10.htm

Diagnosing Java Code: Java Generics Without the Pain, Part 1 by Eric Allen.
Available at
http://www-106.ibm.com/developerworks/java/library/j-djc02113.html

Java Programming (with Passion!)
Available at
http://www.javapassion.com/portal/index.php?option=com_content&view=article&id=66

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

NAME:	Abenes, Enrico O.	**DATE:**	July 11, 2023
COURSE:	CC1 – INTL 1	**SCHEDULE:**	1:30 pm – 5:20 pm MT

Activity Number:

Activity Type: Laboratory Activity

TITLE: Loops

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Identify available loop control structures in Java.

2. Differentiate the different loop control structures in Java.

3. Implement these different loop control structures for a given problem.

4. Create a complete Java program that simulates these basic operations.

INSTRUCTIONS:

1. Make sure you have your own individual account/monitor.

2. Always save your work and log off when not using the computer.

3. By now you should have been familiarized with using your text editor/net beans.

4. By now you should know how to create, save, compile, execute, and debug programs in Java.

5. Use the skills and learning obtained in Midterms for you to successfully finish the learning objectives of this module.

DURATION: One to two Meetings

HANDS-ON:

1. Log on using your own individual account. Use your own username and

password.

2. Open your text editor/NetBeans.

3. Write your next Java program:

```
/* Programmed by: <write your name here>
   Program title: Sum1.java
   Program Date: <write the date today here> */

import java.io.*;
public class Sum1{
  public static void main(String[] args){
      int start = 0, end = 0, sum;
      String input = " ";

      BufferedReader in = new BufferedReader(new
                            InputStreamReader(System.in));

      try{
        System.out.print("Input starting number: ");
        input = in.readLine();
        start = Integer.parseInt(input);

        System.out.print("Input ending number: ");
        input = in.readLine();
        end = Integer.parseInt(input);
      }catch(IOException e){
        System.out.println("Error!");
      }

      if(start >= end){
        System.out.print("Starting number should be lesser ");
        System.out.println("than the ending number. ");
        System.out.println("Try again.");
        System.exit(0);
      }

      if(start%2 == 0)
      {
            start = start + 1;
      }

      sum = 0;
      while(start <= end)
      {
            sum = sum + start;
            start = start + 2;
      }
      System.out.println("Sum = " + sum);
  }
}
```

3.1. Write your next program by copying the source code shown below to your text editor.

3.2. Save the program as Sum1.java then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Input starting number: 3

Input ending number: 10

Sum = 24
```

4. Now let's try another implementation. Copy the source code below and save it as Sum2.java

```
/* Programmed by: <write your name here>
   Program title: Sum2.java
   Program Date: <write the date today here> */

import java.io.*;
public class Sum2{
  public static void main(String[] args){
      int start = 0, end = 0, sum;
      String input = " ";

      BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));
      try{
        System.out.print("Input starting number: ");
        input = in.readLine();
        start = Integer.parseInt(input);

        System.out.print("Input ending number: ");
        input = in.readLine();
        end = Integer.parseInt(input);
      }catch(IOException e){
        System.out.println("Error!");
      }

      if(start >= end){
        System.out.print("Starting number should be lesser ");
        System.out.println("than the ending number. ");
        System.out.println("Try again.");
        System.exit(0);
      }
      if(start%2 == 0){
            start = start + 1;
      }
      sum = 0;
      for(start = start; start <= end; start = start + 2){
            sum = sum + start;
      }
      System.out.println("Sum = " + sum);
  }
}
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

4.1. Simulate and write what will be displayed on the screen. Try using the same input values in your Sum1.java

```
Input starting number: 3

Input ending number: 10

Sum = 24
```

5. Let us try another implementation. Copy the source code and save it as Sum3.java

```
/* Programmed by: <write your name here>
   Program title: Sum3.java
   Program Date: <write the date today here> */

import java.io.*;
public class Sum3{
  public static void main(String[] args){
      int start = 0, end = 0, sum;
      String input = " ";

      BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));
      try{
        System.out.print("Input starting number: ");
        input = in.readLine();
        start = Integer.parseInt(input);
        System.out.print("Input ending number: ");
        input = in.readLine();
        end = Integer.parseInt(input);
      }catch(IOException e){
        System.out.println("Error!");
      }
      if(start >= end){
        System.out.print("Starting number should be lesser ");
        System.out.println("than the ending number. ");
        System.out.println("Try again.");
        System.exit(0);
      }
      if(start%2 == 0){
            start = start + 1;
      }
      sum = 0;
      do{
            sum = sum + start;
            start = start + 2;
      }while(start <= end);

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

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

5.1. Simulate and write what will be displayed on the screen. Try using the same input values in your Sum1.java and Sum2.java

```
Input starting number: 3

Input ending number: 10

Sum = 24
```

6. After your simulation of the three programs, what do you think is the main objective of these programs?

Calculating the total of all odd numbers in a certain range is the main goal of the three Java files. To do this, they gather user input for the range's starting and ending numbers and conduct checks to confirm the accuracy of the information. After that, they iterate across the range, adding each odd number to the total, using various looping constructions (while loop, for loop, and do-while loop). All three programs attempt to compute the sum of odd numbers inside the given range, although employing various looping algorithms.

7. Identify what are the different loop control structures in Java.

for loop – It enables you to use an initialization phrase, condition, and update statement with a code block to run it again for a predetermined number of times.

while loop – If a certain condition is true, a block of code is repeatedly executed.

do-while loop – The only difference between it and the while loop is that it checks the condition at the end to make sure the loop runs at least once.

8. What are the usual similar components of these loop constructs?

Loop Condition – Before beginning each iteration, the condition is examined to determine whether the loop should be continued or ended.

Loop Body – As long as the loop condition is true, this particular piece of code will be run endlessly.

Loop Initialization – Before the loop starts, it requires initializing or setting up any necessary conditions or variables. It establishes the starting variables or conditions needed for the loop to begin.

Loop Update – It specifies how to update or increase the loop control variable after each iteration. At the conclusion of each iteration, it describes the procedure to update the loop control variable.

Loop Entry/Exit – Depending on the circumstance, it specifies how the loop is entered and exits.

9. What do you think is different among these identified loop constructs?

Justify your answer.

Java's **for, while, and do-while loops** are distinct from one another in terms of their structure, loop entry behavior, loop control, and usual usage.

The **for loop** has three distinct statements: initialization, condition, and update. It employs a loop control variable, verifies the condition before each iteration, and enters the loop after setup.

The **while loop**'s structure is more adaptable since it checks the condition before going into the loop body. It is dependent on an outside circumstance and continues as long as it holds true.

Although it also has a flexible structure, the **do-while loop** performs the loop body first to guarantee that it is run at least once. After each iteration, the condition is checked.

10. Create a complete Java program that shall allow the user to accept three integer values representing the START, END, and STEP respectively. The START should always be lesser than the END value, and the STEP is always greater than zero. The program shall print vertically the values starting from the START to the last value which can be equal to or lesser than the END incremented by the STEP value.

Example Output 1: if START = 1, END = 10, STEP = 2

```
Input START value    = 1
Input END value      = 10
Input STEP value     = 2

1
3
5
7
9

Do you want to try again (Y/N)? Y
```

```
Input START value    = -10
Input END value      = 20
Input STEP value     = 5

-10
-5
0
5
10
15
20

Do you want to try again (Y/N)? N
```

Example Output 2: if START = -10, END = 20, STEP = 5

10.1. Your program should automatically detect any errors in the initial input.

10.2. Your program should have an additional feature that asks the user whether the user wants TO TRY AGAIN. The Program will not terminate until the user inputs any character other than 'Y' or 'y'.

10.3. Implement the program using all looping constructs identified earlier.

10.4. Write your complete programs in the space provided.

WHILE LOOP:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: WhileLoop.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class WhileLoop {
    public static void main(String[] args) {
        int start, end, step;
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        while (tryAgain) {
            try {
                System.out.print("Input START value   = ");
                choice = basa.readLine();
                start = Integer.parseInt(choice);

                System.out.print("Input END value     = ");
                choice = basa.readLine();
                end = Integer.parseInt(choice);

                System.out.print("Input STEP value    = ");
                choice = basa.readLine();
                step = Integer.parseInt(choice);

                System.out.println();
                int i = start;
                while (i <= end) {
                    System.out.println(i);
                    i += step;
                }

                System.out.println();
                System.out.print("Do you want to try again (Y/N)? ");
                choice = basa.readLine().toUpperCase();
                tryAgain = choice.equals("Y");
            } catch (IOException | NumberFormatException e) {
                System.out.println("Error!");
                tryAgain = true;
            }
        }
    }
}
```

FOR LOOP:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: ForLoop.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class ForLoop {
    public static void main(String[] args) {
        int start, end, step;
        String choice;
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        for (choice = "Y"; choice.equals("Y"); ) {
            try  {
                System.out.print("Input START value   = ");
                choice = basa.readLine();
                start = Integer.parseInt(choice);

                System.out.print("Input END value     = ");
                choice = basa.readLine();
                end = Integer.parseInt(choice);

                System.out.print("Input STEP value    = ");
                choice = basa.readLine();
                step = Integer.parseInt(choice);

                System.out.println();
                for (int i = start; i <= end; i+= step) {
                    System.out.println(i);
                }

                System.out.println();
                System.out.print("Do you want to try again (Y/N)? ");
                choice = basa.readLine().toUpperCase();
            } catch (IOException | NumberFormatException e) {
                System.out.println("Error!");
                choice = "Y";
            }
        }
    }
}
```

DO-WHILE LOOP:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: DoWhileLoop.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class DoWhileLoop {
    public static void main(String[] args) {
        int start, end, step;
        String choice;
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        boolean tryAgain;
        do {
            try {
                System.out.print("Input START value   = ");
                choice = basa.readLine();
                start = Integer.parseInt(choice);

                System.out.print("Input END value     = ");
                choice = basa.readLine();
                end = Integer.parseInt(choice);

                System.out.print("Input STEP value    = ");
                choice = basa.readLine();
                step = Integer.parseInt(choice);

                System.out.println();
                int i = start;
                do {
                    System.out.println(i);
                    i += step;
                } while (i <= end);

                System.out.println();
                System.out.print("Do you want to try again (Y/N)? ");
                choice = basa.readLine().toUpperCase();
                tryAgain = choice.equals("Y");
            } catch (IOException | NumberFormatException e) {
                System.out.println("Error!");
                tryAgain = true;
            }
        } while (tryAgain);
    }
}
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

NAME:	Abenes, Enrico O.	**DATE:**	July 11, 2023
COURSE:	CC1 – INTL 1	**SCHEDULE:**	1:30 pm – 5:20 pm MT

Activity Number: 2

Activity Type: Laboratory Activity

TITLE: One-Dimensional Array

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Declare a one-dimensional array in Java.

2. Initialize and reference elements in a one-dimensional array.

3. Perform basic operations available for a one-dimensional array.

4. Create a complete Java program that simulates the application of a one-dimensional array.

INSTRUCTIONS:

1. Make sure you have your own individual account.

2. Always keep your account secret from others to avoid unauthorized access to your files.

3. Always save your work and log off when not using the computer.

4. By now you should have been familiarized with using your text editor.

5. By now you should know how to create, save, compile, execute, and debug programs in Java.

DURATION: Two to Three Meetings

HANDS-ON:

1. Log on using your own individual account. Use your own username and password.

2. Open your text editor.

3. Write your next Java program:

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

3.1. Write your next program by copying the source code shown below to your text editor.

```
/* Programmed by: <write your name here>
   Program title: Count.java
   Program Date: <write the date today here> */

import java.io.*;
public class Count{
  public static void main(String[] args){
   int i, n, ctr;
   String input = " ";
   ctr = 0;

   BufferedReader in = new  BufferedReader(new
                           InputStreamReader(System.in));

   for(i = 1; i<=10; i++){
      System.out.print("Input integer number: ");
      try{
        input = in.readLine();
      }catch(IOException e){
        System.out.println("Error!");
      }

      n = Integer.parseInt(input);

      if(n >=0){
       ctr = ctr + 1;
       }
   }
   System.out.println("The total value for counter is " + ctr);
  }
}
```

3.2. Save your program as Count.java then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Input integer number: 32
Input integer number: 42
Input integer number: 32
Input integer number: 54
Input integer number: 34
Input integer number: 42
Input integer number: 42
Input integer number: 43
Input integer number: 65
Input integer number: 43
The total value for counter is 10
```

3.5. What do you think is being counted in the program?

The computer program is keeping track of how many positive integer inputs the user has provided. (quantity)

---

4. Revise your Count.java program and save it as Count2.java.

Thereafter, the user will be asked to input 10 integer numbers and display how many of these 10 numbers are even and how many are odd. Use only one variable for your counter.

4.1. Write your complete Java program here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: Count2.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class Count2 {
    public static void main(String[] args) {
        int i, n, ctr;
        String input = " ";
        ctr = 0;

        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
```

```
        for (i = 1; i <= 10; i++) {
            System.out.print("Input integer number: ");
            try {
                input = in.readLine();
            } catch (IOException e) {
                System.out.println("Error!");
            }

            n = Integer.parseInt(input);

            if (n % 2 == 0) {
                ctr++;
            }
        }
        System.out.println("Number of even integers: " + ctr);
        System.out.println("Number of odd integers: " + (10 - ctr));
    }
}
```

4.3. Run your program.

4.4. Simulate and write what will be displayed on the screen.

```
Input integer number: 1
Input integer number: 2
Input integer number: 3
Input integer number: 6
Input integer number: 7
Input integer number: 9
Input integer number: 12
Input integer number: 13
Input integer number: 14
Input integer number: 17
Number of even integers: 4
Number of odd integers: 6
```

4.5. What do you think is the purpose of counters?

A **counter variable** is a variable that counts the instances at which a particular line of code is run.

5. Write your next program by copying the source code shown below to your text editor.

5.1. Create a new program and save it as Accum.java

```
/* Programmed by: <write your name here>
   Program title: Accum.java
   Program Date: <write the date today here> */

public class Accum{
  public static void main(String[] args){
      int i, n, sum = 0;
      String input = " ";
       BufferedReader in = new  BufferedReader(new
                           InputStreamReader(System.in));

      for(i = 0; i<10; i++){
        System.out.print("Input integer number: ");
        try{
          input = in.readLine();
        }catch(IOException e){
          System.out.println("Error!");
        }

        n = Integer.parseInt(input);
       sum = sum + n;
      }
      System.out.println("The sum of the integers is " + sum);
  }
}
```

5.2. Save and then compile your program until no errors and warnings are reported.

5.3. Run your program.

5.4. Simulate and write what will be displayed on the screen

```
Input integer number: 12
Input integer number: 54
Input integer number: 63
Input integer number: 45
Input integer number: 76
Input integer number: 98
Input integer number: 87
Input integer number: 65
Input integer number: 46
Input integer number: 74
The sum of the integers is 620
```

5.5. What do you think is being accumulated in your program?

The user-inputted sum of 10 integer numbers is being added up by the application. It asks for an integer ten times from the user, reads the input, and turns it into an integer. It displays the **user-inputted integers' combined total.**

6. Revise your Accum.java program and save it as Accum2.java.

Thereafter, the user will be asked to input 10 integer numbers and display the sum of all even numbers and the sum of all odd numbers.

6.1. Write your complete Java program here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: Accum2.java
   Program Date: July 10, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```

```
public class Accum2 {
    public static void main(String[] args) {
        int i, n, sum_even = 0, sum_odd = 0;
        String input = " ";

        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));

        for (i = 0; i < 10; i++) {
            System.out.print("Input integer number: ");
            try {
                input = in.readLine();
            } catch (IOException e) {
                System.out.println("Error!");
            }

            n = Integer.parseInt(input);

            if (n % 2 == 0) {
                sum_even += n;
            } else {
                sum_odd += n;
            }
        }
        System.out.println("The sum of even integers is: " + sum_even);
        System.out.println("The sum of odd integers is: " + sum_odd);
    }
}
```

6.3. Run your program.

6.4. Simulate and write what will be displayed on the screen.

```
Input integer number: 54
Input integer number: 76
Input integer number: 54
Input integer number: 68
Input integer number: 97
Input integer number: 65
Input integer number: 43
Input integer number: 73
Input integer number: 67
Input integer number: 64
The sum of even integers is: 316
The sum of odd integers is: 345
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

6.5. What do you think is the purpose of accumulators?

The results of successive additions are partially stored in an **accumulator variable.** Usually, the value that is added comes from another variable.

7. Differentiate new counters from accumulators.

An **accumulator** is used to gather data and compute a cumulative value over numerous operations, whereas a **counter** is used to keep track of the quantity of occurrences or iterations.

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

NAME:	Abenes, Enrico O.	**DATE:**	July 11, 2023
COURSE:	CC1 – INTL 1	**SCHEDULE:**	1:30 pm – 5:20 pm MT

Activity Number: 3

Activity Type: Laboratory Activity

TITLE: One-Dimensional Array

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Declare a one-dimensional array in Java.

2. Initialize and reference elements in a one-dimensional array.

3. Perform basic operations available for a one-dimensional array.

4. Create a complete Java program that simulates the application of a one-dimensional array.

INSTRUCTIONS:

1. Make sure you have your own individual account.

2. Always keep your account secret from others to avoid unauthorized access to your files.

3. Always save your work and log off when not using the computer.

4. By now you should have been familiarized with using your text editor.

5. By now you should know how to create, save, compile, execute, and debug programs in Java.

DURATION: Two to Three Meetings

HANDS-ON:

1. Log on using your own individual account. Use your own username and password.

2. Open your text editor.

3. Write your next Java program:

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

3.1. Write your next program by copying the source code shown below to your text editor.

```
/* Programmed by: <write your name here>
   Program title: List.java
   Program Date: <write the date today here> */

import java.io.*;
public class List{
  public static void main(String[] args){
    int list[] = new int[10];
    int i, num = 0;
    String input = " ";

    BufferedReader in = new BufferedReader(new
                            InputStreamReader(System.in));

    for(i = 0; i < 10; i++){
      list[i] = 0;
    }
    for(i = 0; i < 10; i++){
      System.out.print("Input value for list[" + i +
                                "] = ");
      try{
       input = in.readLine();
      }catch(IOException e){}
      num = Integer.parseInt(input);
      list[i] = num;
     }

     for(i = 0; i < 10; i++){
       System.out.println("list[" + i + "] = " +list[i]);
     }
  }
}
```

3.2. Save the program as List.java then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Input value for list [0) = 3
Input value for list [1) = 5
Input value for list [2) = 5
Input value for list [3) = 7
Input value for list [4) = 5
Input value for list [5) = 2
Input value for list [6) = 6
Input value for list [7) = 9
Input value for list [8) = 6
Input value for list [9) = 7
list [0] = 3
list [1] = 5
list [2] = 5
list [3] = 7
list [4] = 5
list [5] = 2
list [6] = 6
list [7] = 9
list [8] = 6
list [9] = 7
```

3.5. What do you think is the purpose of the first for-loop?

Before the user enters any specific values, the **first for-loop** initializes the array elements to 0, giving each element a starting point value.

3.6. What do you think is the purpose of the second for-loop?

The user can input values for each element of the list array using the **second for-loop**, giving the application user-specific data to work with afterwards.

3.7. What do you think is the purpose of the third for-loop?

The **third for-loop** is in charge of reporting the list array's values to the console so you can check the values the user supplied and make sure they were properly stored in the array.

3.8. How many elements can your list contain?

The fixed size of the list array in the provided code is 10, therefore it may hold **10 elements**. We can modify the array declaration's size to change how many elements the array can hold.

3.9. What index value is the first element located?

The first entry in the context of the list array in the code is at **index 0**.

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

3.10. Given the <size> of the list, at what index is the last element located?

The list's fixed length is 10 items. Consequently, **index 9** is where the final element can be found.

---

4. Create a new program and save it as List2.java. Thereafter, create 3 lists and name is as list1, list2, and list3, having 10 elements each. The user should input 10 integer values for list1 and 10 integer values for list2. Your program should add the contents of list1 and list2 then store the sum to list3. Your program should display horizontally the values of list1, list2, and list3. Use loops.

Example Output:

```
List1 : 1  3  2  5  7  8  5  6  9  4
List2 : 2  1  4  3  2  1  4  2  0  2
List3 : 3  4  6  8  9  9  9  8  9  6
```

Hint: use for -loops!
list3[0] = list1[0] + list2[0]
list3[1] = list1[1] + list2[1]
list3[2] = list1[2] + list2[2]
list3[3] = list1[3] + list2[3]
list3[4] = list1[4] + list2[4]
list3[5] = list1[5] + list2[5]
list3[6] = list1[6] + list2[6]
list3[7] = list1[7] + list2[7]
list3[8] = list1[8] + list2[8]
list3[9] = list1[9] + list2[9]

4.1. Write your complete Java program here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: List2.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class List2 {
    public static void main(String[] args) {
        String input = " ";
```

```
        int[] list1 = new int[10];
        int[] list2 = new int[10];
        int[] list3 = new int[10];

        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        try {
            System.out.println("Enter Ten(10) Integer Values for
List1");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list1[" + i + "] =
");
                input = basa.readLine();
                list1[i] = Integer.parseInt(input);
            }

            System.out.println(" ");
            System.out.println("Enter Ten(10) Integer Values for
List2");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list2[" + i + "] =
");
                input = basa.readLine();
                list2[i] = Integer.parseInt(input);
            }

            System.out.print("List1 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list1[i] + "  ");
            }
            System.out.println();

            System.out.print("List2 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list2[i] + "  ");
            }
            System.out.println();

            for (int i = 0; i < 10; i++) {
                list3[i] = list1[i] + list2[i];
            }

            System.out.print("List3 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list3[i] + "  ");
            }
            System.out.println();

        } catch (IOException e) {
            System.out.println("Error!");
        }
    }
```

4.2. Save then compile your program.

4.3. Run your program.

4.4. Simulate and write what will be displayed on the screen.

```
Enter Ten(10) Integer Values for List1
Input value for list1[0] = 1
Input value for list1[1] = 3
Input value for list1[2] = 4
Input value for list1[3] = 2
Input value for list1[4] = 1
Input value for list1[5] = 5
Input value for list1[6] = 4
Input value for list1[7] = 2
Input value for list1[8] = 4
Input value for list1[9] = 2

Enter Ten(10) Integer Values for List2
Input value for list2[0] = 5
Input value for list2[1] = 3
Input value for list2[2] = 2
Input value for list2[3] = 5
Input value for list2[4] = 1
Input value for list2[5] = 4
Input value for list2[6] = 3
Input value for list2[7] = 5
Input value for list2[8] = 2
Input value for list2[9] = 1
List1 : 1  3  4  2  1  5  4  2  4  2
List2 : 5  3  2  5  1  4  3  5  2  1
List3 : 6  6  6  7  2  9  7  7  6  3
```

5. Revise your List2.java program. Thereafter, your program should display the highest value in list3 and the lowest value in list3.

5.1. Write the additional codes here:

```
int mataas = list3[0];
int mababa = list3[0];

for (int i = 1; i < 10; i++) {
    if (list3[i] > mataas) {
        mataas = list3[i];
    }
    if (list3[i] < mababa) {
        mababa = list3[i];
    }
}
```

```
System.out.println("Highest value in List3: " + mataas);
System.out.println("Lowest value in List3: " + mababa);
```

6. Revise your List2.java program. Thereafter, your program should allow the user to input an integer value to be searched in list3. Your program should display whether the inputted integer value is found in list3, how many of them are in list3, and what are their locations in list3.

Example output:

```
List1 : 1  3  2  5  7  8  5  6  9  4
List2 : 2  1  4  3  2  1  4  2  0  2
List3 : 3  4  6  8  9  9  9  8  9  6

Input value to search in List3: 9

The value 9 is in List3!
There are 4 of it in List3.
Located at: list3[4], list3[5], list3[6], list3[8]
```

6.1. Save and then compile your program until no errors and warnings are reported.

6.2. Run your program.

6.3. Simulate and write what will be displayed on the screen.

```
Enter Ten(10) Integer Values for List1
Input value for list1[0] = 1
Input value for list1[1] = 4
Input value for list1[2] = 5
Input value for list1[3] = 3
Input value for list1[4] = 2
Input value for list1[5] = 1
Input value for list1[6] = 5
Input value for list1[7] = 4
Input value for list1[8] = 5
Input value for list1[9] = 3
```

```
Enter Ten(10) Integer Values for List2
Input value for list2[0] = 1
Input value for list2[1] = 4
Input value for list2[2] = 2
Input value for list2[3] = 5
Input value for list2[4] = 4
Input value for list2[5] = 2
Input value for list2[6] = 1
Input value for list2[7] = 4
Input value for list2[8] = 3
Input value for list2[9] = 4

List1 : 1  4  5  3  2  1  5  4  5  3
List2 : 1  4  2  5  4  2  1  4  3  4
List3 : 2  8  7  8  6  3  6  8  8  7

Input value to search in List3: 8

The value 8 is in List3!
There are 4 of it in List 3.
Located at: list3[1]. list3[3]. list3[7]. list3[8]
```

6.4. Write your complete source code here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: List2.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class List2 {
    public static void main(String[] args) {
        String search_input, input;
        boolean hanap = false;
        int search_value, ctr = 0;

        int[] list1 = new int[10];
        int[] list2 = new int[10];
        int[] list3 = new int[10];

        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        try {
            System.out.println("Enter Ten(10) Integer Values for
List1");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list1[" + i + "] =
");
                input = basa.readLine();
                list1[i] = Integer.parseInt(input);
            }

            System.out.println(" ");
            System.out.println("Enter Ten(10) Integer Values for
List2");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list2[" + i + "] =
");
                input = basa.readLine();
                list2[i] = Integer.parseInt(input);
            }

            System.out.println(" ");
            System.out.print("List1 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list1[i] + "  ");
            }
            System.out.println();

            System.out.print("List2 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list2[i] + "  ");
            }
            System.out.println();

            for (int i = 0; i < 10; i++) {
                list3[i] = list1[i] + list2[i];
            }

            System.out.print("List3 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list3[i] + "  ");
            }
            System.out.println();

            System.out.println(" ");
            System.out.print("Input value to search in List3: ");
            search_input = basa.readLine();
            search_value = Integer.parseInt(search_input);

            StringBuilder lokasyon = new StringBuilder();
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
            System.out.println(" ");
            for (int i = 0; i < 10; i++) {
                if (list3[i] == search_value) {
                    if (!hanap) {
                        System.out.println("The value " + search_value
+ " is in List3!");
                        hanap = true;
                    }
                    ctr++;
                    lokasyon.append("list3[").append(i).append("]. ");
                }
            }

            if (hanap) {
                System.out.println("There are " + ctr + " of it in List
3.");
                System.out.println("Located at: " +
lokasyon.substring(0, lokasyon.length() - 2));
            } else {
                System.out.println("The value " + search_value + " is
not found in List3");
            }

        } catch (IOException e) {
            System.out.println("Error!");
        }
    }
}
```

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

NAME:	Abenes, Enrico O.	**DATE:**	July 11, 2023
COURSE:	CC1 – INTL 1	**SCHEDULE:**	1:30 pm – 5:20 pm MT

Activity Number: 4

Activity Type: Laboratory Activity

TITLE: Java Methods

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Identify the types of methods in the Java programming language.

2. Identify what portions of the main program can be subdivided into sub-programs.

3. Create methods or sub-programs that can be called in the main program.

4. Distinguish the different parameters used in creating methods.

5. Create a complete Java program that utilizes methods.

INSTRUCTIONS:

1. Make sure you have your own individual account.

2. Always keep your account secret from others to avoid unauthorized access to your files.

3. Always save your work and log off when not using the computer.

4. By now you should have been familiarized with using your text editor.

5. By now you should know how to create, save, compile, execute, and debug programs in Java.

DURATION: Two to Three Meetings

HANDS-ON:

1. Log on using your own individual account. Use your own username and password.

2. Open your text editor.

3. Write your next Java program:

    3.1. Write your next program by copying the source code shown below to your text editor.

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
/* Programmed by: <write your name here>
   Program title: PrintHello.java
   Program Date: <write the date today here> */

public class PrintHello{
     public static void printHello(){
          System.out.print("HELLO");
     }
     public static void printSpace(){
          System.out.print(" ");
     }
     public static void printWorld(){
          System.out.println("WORLD!");
     }
     public static void allTogetherNow(){
          printHello();
          printSpace();
          printWorld();
     }

     public static void main(String[] args){
          AllTogetherNow();
     }
}
```

3.2. Save the program as PrintHello.java then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Hello WORLD!
```

3.5. Enumerate all user-defined methods:

printHello() – "Hello" is printed to the console using this way.

printSpace() – This technique prints the character "space" to the console.

printWorld() – "WORLD!" is printed to the console by this way.

allTogetherNow() – The printHello(), printSpace(), and printWorld() methods are sequentially invoked by this method.

3.6. Enumerate all pre-defined methods:

System,out print() – Text can be printed on the console using this technique.

String[] args – This is the input for the main method, a built-in method that acts as the starting point for Java programs.

> ***HINT:*** To call a method, indicate the name of the method preceded by the **()** symbol. User-define methods are methods you created , while pre-defined functions are functions already inherent to the compiler.

4. Is the method parameter always void? No

```
public class OneParam{
     public static void oneParam(int x){
          System.out.println("Number is " + x);
     }

     public static void main(String[] args){
          int x;
          x = 100;
          oneParam(x);
     }
}
```

4.1 To answer that, let's simulate the following program

4.2 Save your program as OneParam.java

4.3 Run and write what will be displayed on the screen.

```
Number is 100
```

4.4. Insert the statement oneParam(75);

What can you conclude? That is, is it possible to use a constant value as a parameter when a method is called?

When you wish to send a method a certain value that is known at compile time and doesn't need to be dynamically computed during runtime, using constant values as method parameters **might be helpful.**

4.5 Declare another integer variable in your main method as follows:

int y; Initialize the variable y to 150. Thereafter, write another statement OneParam(y);

What can you conclude? That is, is it possible to use a variable name that is not the same as that used in the method definition?

Using a variable name other than the one given in the method description is technically **conceivable**. As long as the types of the arguments and parameters match, various variable names can be used to pass values when invoking a method.

4.6 Insert the statement OneParam(x+y);

What can you conclude? That is, is it possible to use an expression (that will evaluate an integer value) as a parameter when the method is called?

When invoking a method, it is **possible** to pass an expression that evaluates to an integer value as a parameter.

4.7 Insert the statement OneParam();

What can you conclude? That is, can you call a method requiring a parameter with a passing one? What error is reported if any?

A method that requires a parameter **cannot be called** without one being sent in. There is a **compilation problem** when OneParam() is called without any parameters.

5. Let us create another program and save it as TwoParams.java

5.1. Copy the source code

```
public class TwoParams{
  public static void twoParams(int x, int y){
     System.out.println("First parameter is " + x);
     System.out.println("Second parameter is " + y);
  }

  public static void main(String[] args){
     int x = 5;
     int y = 10;

     int a;
     int b;

     twoParams(10, 20);
     twoparams(x + 2, y * 10);

     a = -2;
     b = 22;
     twoParams(a, b);
  }
}
```

5.2. Save and then compile your program until no errors and warnings are reported.

5.3. Run your program.

5.4. Simulate and write what will be displayed on the screen.

```
First parameter is 10
Second parameter is 20
First parameter is 7
Second parameter is 100
First parameter is -2
Second parameter is 22
```

5.5. Insert the statement twoParams(y, x);

What can you conclude? Is the result the same as of twoParams(x,y)?

Both "First parameter is 10" and "Second parameter is 5" will be printed. TwoParams(x, y); prints "First parameter is 5" and "Second parameter is 10", however this outcome **is different.**

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

5.5.1 What will happen if we forgot to type the comma between parameter? For example, will twoParams(x y) work?

Java will encounter a **syntax error** as a result. The code won't successfully compile.

5.6. What will happen if we only supplied one parameter? For example, will twoParams(x) work? No, compilation error.

5.7. What will happen if we forgot all the parameters?

For example, will twoParams() work? No, compilation error.

5.8. What will happen if we used more than two parameters? For example, will twoParams(x, y, a) work? No, compilation error.

6. Are all methods of return type void? No

6.1. To answer that let's simulate the following program:

```
//Return value

public class WithReturnValue{
  public static int printSum(int x, int y){
     z = x + y;
     return z;
  }

  public static void main(String[] args){
     int a, b, c;

     System.out.println("Sum = " + printSum(5, 10));
     c = printSum(100, 300);
     System.out.println("c = " + c);

     a = 25;
     b = 75;

     System.out.println("Sum = " + printSum(a, b));
     System.out.println("Sum = " + Sum(a+2, b - 3));

  }
}
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

6.2. Save your program as WithReturnValue.java then compile and run and simulate your program. Write what will be displayed on the screen:

```
Compilation Error

Command execution failed.

BUILD FAILURE
```

6.3. What will happen if the statement return z; was omitted?

When the code attempts to delete the method's return value, a **compilation error** will still result.

6.4. Write a method named printDifference that accepts two integer parameters named x and y then computes the difference (x – y). Call this method inside the main().

```
/* Programmed by: Abenes, Enrico O.
   Program Title: Difference.java
   Program Date: July 11, 2023*/

package intl.cc1;

public class Difference {
    public static int printDifference(int x, int y) {
        int negatibo = x - y;
        return negatibo;
    }

    public static void main(String[] args) {
        int a = 10;
        int b = 5;

        int resulta = printDifference(a, b);
        System.out.println("Difference = " + resulta);
    }
}
```

6.5. Write a method named printProduct that accepts two double-type parameters named x and y. The method should compute and return the value of the product of x and y. Use double as return type. Call this method inside main();

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
/* Programmed by: Abenes, Enrico O.
   Program Title: Product.java
   Program Date: July 11, 2023*/

package intl.cc1;

public class Product {
    public static double printProduct(double x, double y) {
        double producto = x * y;
        return producto;
    }

    public static void main(String[] args) {
        double a = 4.6;
        double b = 5.9;

        double resulta = printProduct(a, b);
        System.out.println("Product = " + resulta);
    }
}
```

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

NAME:	Abenes, Enrico O.	**DATE:**	July 11, 2023
COURSE:	CC1 – INTL 1	**SCHEDULE:**	1:30 pm – 5:20 pm MT

Activity Number:

Activity Type: Laboratory Activity

TITLE: Loops

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Identify available loop control structures in Java.

2. Differentiate the different loop control structures in Java.

3. Implement these different loop control structures for a given problem.

4. Create a complete Java program that simulates these basic operations.

INSTRUCTIONS:

1. Make sure you have your own individual account/monitor.

2. Always save your work and log off when not using the computer.

3. By now you should have been familiarized with using your text editor/net beans.

4. By now you should know how to create, save, compile, execute, and debug programs in Java.

5. Use the skills and learning obtained in Midterms for you to successfully finish the learning objectives of this module.

DURATION: One to two Meetings

HANDS-ON:

1. Log on using your own individual account. Use your own username and

password.

2. Open your text editor/NetBeans.

3. Write your next Java program:

```
/* Programmed by: <write your name here>
   Program title: Sum1.java
   Program Date: <write the date today here> */

import java.io.*;
public class Sum1{
  public static void main(String[] args){
      int start = 0, end = 0, sum;
      String input = " ";

      BufferedReader in = new BufferedReader(new
                            InputStreamReader(System.in));

      try{
        System.out.print("Input starting number: ");
        input = in.readLine();
        start = Integer.parseInt(input);

        System.out.print("Input ending number: ");
        input = in.readLine();
        end = Integer.parseInt(input);
      }catch(IOException e){
        System.out.println("Error!");
      }

      if(start >= end){
        System.out.print("Starting number should be lesser ");
        System.out.println("than the ending number. ");
        System.out.println("Try again.");
        System.exit(0);
      }

      if(start%2 == 0)
      {
            start = start + 1;
      }

      sum = 0;
      while(start <= end)
      {
            sum = sum + start;
            start = start + 2;
      }
      System.out.println("Sum = " + sum);
  }
}
```

3.1. Write your next program by copying the source code shown below to your text editor.

3.2. Save the program as Sum1.java then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Input starting number: 3

Input ending number: 10

Sum = 24
```

4. Now let's try another implementation. Copy the source code below and save it as Sum2.java

```
/* Programmed by: <write your name here>
   Program title: Sum2.java
   Program Date: <write the date today here> */

import java.io.*;
public class Sum2{
  public static void main(String[] args){
      int start = 0, end = 0, sum;
      String input = " ";

      BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));
      try{
        System.out.print("Input starting number: ");
        input = in.readLine();
        start = Integer.parseInt(input);

        System.out.print("Input ending number: ");
        input = in.readLine();
        end = Integer.parseInt(input);
      }catch(IOException e){
        System.out.println("Error!");
      }

      if(start >= end){
        System.out.print("Starting number should be lesser ");
        System.out.println("than the ending number. ");
        System.out.println("Try again.");
        System.exit(0);
      }
      if(start%2 == 0){
            start = start + 1;
      }
      sum = 0;
      for(start = start; start <= end; start = start + 2){
            sum = sum + start;
      }
      System.out.println("Sum = " + sum);
  }
}
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

4.1. Simulate and write what will be displayed on the screen. Try using the same input values in your Sum1.java

```
Input starting number: 3

Input ending number: 10

Sum = 24
```

5. Let us try another implementation. Copy the source code and save it as Sum3.java

```
/* Programmed by: <write your name here>
   Program title: Sum3.java
   Program Date: <write the date today here> */

import java.io.*;
public class Sum3{
  public static void main(String[] args){
      int start = 0, end = 0, sum;
      String input = " ";

      BufferedReader in = new BufferedReader(new
                          InputStreamReader(System.in));
      try{
        System.out.print("Input starting number: ");
        input = in.readLine();
        start = Integer.parseInt(input);
        System.out.print("Input ending number: ");
        input = in.readLine();
        end = Integer.parseInt(input);
      }catch(IOException e){
        System.out.println("Error!");
      }
      if(start >= end){
        System.out.print("Starting number should be lesser ");
        System.out.println("than the ending number. ");
        System.out.println("Try again.");
        System.exit(0);
      }
      if(start%2 == 0){
            start = start + 1;
      }
      sum = 0;
      do{
            sum = sum + start;
            start = start + 2;
      }while(start <= end);

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

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

5.1. Simulate and write what will be displayed on the screen. Try using the same input values in your Sum1.java and Sum2.java

```
Input starting number: 3

Input ending number: 10

Sum = 24
```

6. After your simulation of the three programs, what do you think is the main objective of these programs?

Calculating the total of all odd numbers in a certain range is the main goal of the three Java files. To do this, they gather user input for the range's starting and ending numbers and conduct checks to confirm the accuracy of the information. After that, they iterate across the range, adding each number to the total, using various looping constructions (while loop, for loop, and do-while loop). All three programs attempt to compute the sum of numbers inside the given range, although employing various looping algorithms.

7. Identify what are the different loop control structures in Java.

for loop – It enables you to use an initialization phrase, condition, and update statement with a code block to run it again for a predetermined number of times.

while loop – If a certain condition is true, a block of code is repeatedly executed.

do-while loop – The only difference between it and the while loop is that it checks the condition at the end to make sure the loop runs at least once.

8. What are the usual similar components of these loop constructs?

Condition – Before beginning each iteration, the condition is examined to determine whether the loop should be continued or ended.

Body – As long as the loop condition is true, this particular piece of code will be run endlessly.

Initialization – Before the loop starts, it requires initializing or setting up any necessary conditions or variables. It establishes the starting variables or conditions needed for the loop to begin.

Update – It specifies how to update or increase the loop control variable after each iteration. At the conclusion of each iteration, it describes the procedure to update the loop control variable.

Entry/Exit – Depending on the circumstance, it specifies how the loop is entered and exits.

9. What do you think is different among these identified loop constructs?

Justify your answer.

Java's **for, while, and do-while loops** are distinct from one another in terms of their structure, loop entry behavior, loop control, and usual usage.

The **for loop** employs a loop control variable, verifies the condition before each iteration, and enters the loop after setup.

The **while loop**'s structure is more adaptable since it checks the condition before going into the loop body. It is dependent on an outside circumstance and continues as long as it holds true.

Although it also has a flexible structure, the **do-while loop** performs the loop body first to guarantee that it is run at least once. After each iteration, the condition is checked.

10. Create a complete Java program that shall allow the user to accept three integer values representing the START, END, and STEP respectively. The START should always be lesser than the END value, and the STEP is always greater than zero. The program shall print vertically the values starting from the START to the last value which can be equal to or lesser than the END incremented by the STEP value.

Example Output 1: if START = 1, END = 10, STEP = 2

```
Input START value   = 1
Input END value     = 10
Input STEP value    = 2

1
3
5
7
9

Do you want to try again (Y/N)? Y
```

```
Input START value    = -10
Input END value      = 20
Input STEP value     = 5

-10
-5
0
5
10
15
20

Do you want to try again (Y/N)? N
```

Example Output 2: if START = -10, END = 20, STEP = 5

10.1. Your program should automatically detect any errors in the initial input.

10.2. Your program should have an additional feature that asks the user whether the user wants TO TRY AGAIN. The Program will not terminate until the user inputs any character other than 'Y' or 'y'.

10.3. Implement the program using all looping constructs identified earlier.

10.4. Write your complete programs in the space provided.

WHILE LOOP:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: WhileLoop.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class WhileLoop {
    public static void main(String[] args) {
        int start, end, step;
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        while (tryAgain) {
            try {
                System.out.print("Input START value   = ");
                choice = basa.readLine();
                start = Integer.parseInt(choice);

                System.out.print("Input END value     = ");
                choice = basa.readLine();
                end = Integer.parseInt(choice);

                System.out.print("Input STEP value    = ");
                choice = basa.readLine();
                step = Integer.parseInt(choice);

                System.out.println();
                int i = start;
                while (i <= end) {
                    System.out.println(i);
                    i += step;
                }

                System.out.println();
                System.out.print("Do you want to try again (Y/N)? ");
                choice = basa.readLine().toUpperCase();
                tryAgain = choice.equals("Y");
            } catch (IOException | NumberFormatException e) {
                System.out.println("Error!");
                tryAgain = true;
            }
        }
    }
}
```

FOR LOOP:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: ForLoop.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class ForLoop {
    public static void main(String[] args) {
        int start, end, step;
        String choice;
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        for (choice = "Y"; choice.equals("Y"); ) {
            try  {
                System.out.print("Input START value   = ");
                choice = basa.readLine();
                start = Integer.parseInt(choice);

                System.out.print("Input END value     = ");
                choice = basa.readLine();
                end = Integer.parseInt(choice);

                System.out.print("Input STEP value    = ");
                choice = basa.readLine();
                step = Integer.parseInt(choice);

                System.out.println();
                for (int i = start; i <= end; i+= step) {
                    System.out.println(i);
                }

                System.out.println();
                System.out.print("Do you want to try again (Y/N)? ");
                choice = basa.readLine().toUpperCase();
            } catch (IOException | NumberFormatException e) {
                System.out.println("Error!");
                choice = "Y";
            }
        }
    }
}
```

DO-WHILE LOOP:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: DoWhileLoop.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class DoWhileLoop {
    public static void main(String[] args) {
        int start, end, step;
        String choice;
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

```
        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        boolean tryAgain;
        do {
            try {
                System.out.print("Input START value   = ");
                choice = basa.readLine();
                start = Integer.parseInt(choice);

                System.out.print("Input END value     = ");
                choice = basa.readLine();
                end = Integer.parseInt(choice);

                System.out.print("Input STEP value    = ");
                choice = basa.readLine();
                step = Integer.parseInt(choice);

                System.out.println();
                int i = start;
                do {
                    System.out.println(i);
                    i += step;
                } while (i <= end);

                System.out.println();
                System.out.print("Do you want to try again (Y/N)? ");
                choice = basa.readLine().toUpperCase();
                tryAgain = choice.equals("Y");
            } catch (IOException | NumberFormatException e) {
                System.out.println("Error!");
                tryAgain = true;
            }
        } while (tryAgain);
    }
}
```

UNIVERSITY OF THE CORDILLERAS
College of Information Technology and Computer Science
CC1 – Computing Fundamentals
Term & School Year: Third Trimester S.Y. 2022-2023
Faculty: Mailynn K. Sagayo

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**

Final Laboratory Activity 2

Name: Abenes, Enrico O.
Subject Code & Schedule: CC1-INTL1, MT 1:30 pm – 5:20 pm
Course and Year: BSIT – 1st Year

TITLE: One Dimensional Array

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Declare a one-dimensional array in Java.
2. Initialize and reference elements in a one-dimensional array.
3. Perform basic operations available for a one-dimensional array.
4. Create a complete Java program that simulates the application of a one-dimensional array.

INSTRUCTIONS:

1. Make sure you have your own individual account.
2. Always keep your account secret to others to avoid unauthorized access to your files.
3. Always save your work and log-off when not using the computer.
4. By now you should have been familiarized using your text editor.
5. By now you should know how to create, save, compile, execute, and debug programs in Java.
6. Use the skills and learning obtained in Prelim Activity1 to Midterm Activity 4 in order for you to successfully finish the learning objectives of this module.

DURATION: Two to Three Meetings

HANDS-ON:

1. Log-on using your own individual account. Use your own **username** and **password.**
2. Open your text editor.
3. Write your next Java program:
   3.1. Write your next program by copying the source code shown below to your text editor.

```
/* Programmed by: <write your name here>
   Program title: List.java
   Program Date: <write the date today here> */

import java.io.*;
public class List{
  public static void main(String[] args){
    int list[] = new int[10];
    int i, num = 0;
    String input = " ";

    BufferedReader in = new BufferedReader(new
                            InputStreamReader(System.in));

    for(i = 0; i < 10; i++){
      list[i] = 0;
    }
    for(i = 0; i < 10; i++){
      System.out.print("Input value for list[" + i +
                                "] = ");
      try{
       input = in.readLine();
      }catch(IOException e){}
      num = Integer.parseInt(input);
      list[i] = num;
     }

     for(i = 0; i < 10; i++){
       System.out.println("list[" + i + "] = " +list[i]);
     }
  }
}
```

3.2. Save the program as **`List.java`** then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Input value for list [0) = 3
Input value for list [1) = 5
Input value for list [2) = 5
Input value for list [3) = 7
Input value for list [4) = 5
Input value for list [5) = 2
Input value for list [6) = 6
Input value for list [7) = 9
Input value for list [8) = 6
Input value for list [9) = 7
```

```
list [0] = 3
list [1] = 5
list [2] = 5
list [3] = 7
list [4] = 5
list [5] = 2
list [6] = 6
list [7] = 9
list [8] = 6
list [9] = 7
```

1.1. What do you think is the purpose of the first for-loop?

Before the user enters any specific values, the **first for-loop** initializes the array elements to 0, giving each element a starting point value.

1.2. What do you think is the purpose of the second for-loop?

The user can input values for each element of the list array using the **second for-loop**, giving the application user-specific data to work with afterwards.

1.3. What do you think is the purpose of the third for-loop?

The **third for-loop** is in charge of reporting the list array's values to the console so you can check the values the user supplied and make sure they were properly stored in the array.

1.4. How many elements can your list contain? Ten (10) elements

1.5. What index value is the first element located? Index value zero (0)

1.6. Given the <size> of the list, at what index is the last element located? Index value nine (9)

2. Create a new program and save it as **`List2.java`**. Thereafter, create 3 lists and name is as **list1**, **list2**, and **list3**, having 10 elements each. The user should input 10 integer values for **list1** and 10 integer values for **list2**. Your program should add the contents of **list1** and **list2** then store the sum to **list3**. Your program should display horizontally the values of **list1**, **list2**, and **list3**. Use loops.

Example output:

```
List1 : 1  3  2  5  7  8  5  6  9  4
List2 : 2  1  4  3  2  1  4  2  0  2
List3 : 3  4  6  8  9  9  9  8  9  6
```

Hint: use for -loops!
list3[0] = list1[0] + list2[0]
list3[1] = list1[1] + list2[1]
list3[2] = list1[2] + list2[2]
list3[3] = list1[3] + list2[3]
list3[4] = list1[4] + list2[4]
list3[5] = list1[5] + list2[5]
list3[6] = list1[6] + list2[6]
list3[7] = list1[7] + list2[7]
list3[8] = list1[8] + list2[8]
list3[9] = list1[9] + list2[9]

2.1. Write your complete Java program here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: List2.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class List2 {
    public static void main(String[] args) {
        String input = " ";

        int[] list1 = new int[10];
        int[] list2 = new int[10];
        int[] list3 = new int[10];

        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        try {
            System.out.println("Enter Ten(10) Integer Values for
List1");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list1[" + i + "] =
");
                input = basa.readLine();
                list1[i] = Integer.parseInt(input);
            }

            System.out.println(" ");
            System.out.println("Enter Ten(10) Integer Values for
List2");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list2[" + i + "] =
");
                input = basa.readLine();
                list2[i] = Integer.parseInt(input);
            }
```

```
            System.out.print("List1 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list1[i] + "  ");
            }
            System.out.println();

            System.out.print("List2 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list2[i] + "  ");
            }
            System.out.println();

            for (int i = 0; i < 10; i++) {
                list3[i] = list1[i] + list2[i];
            }

            System.out.print("List3 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list3[i] + "  ");
            }
            System.out.println();

        } catch (IOException e) {
            System.out.println("Error!");
        }
    }
}
```

2.2. Save then compile your program until no errors and warnings are reported.
2.3. Run your program.
2.4. Simulate and write what will be displayed on the screen.

```
Enter Ten(10) Integer Values for List1
Input value for list1[0] = 1
Input value for list1[1] = 3
Input value for list1[2] = 4
Input value for list1[3] = 2
Input value for list1[4] = 1
Input value for list1[5] = 5
Input value for list1[6] = 4
Input value for list1[7] = 2
Input value for list1[8] = 4
Input value for list1[9] = 2

Enter Ten(10) Integer Values for List2
Input value for list2[0] = 5
Input value for list2[1] = 3
Input value for list2[2] = 2
Input value for list2[3] = 5
Input value for list2[4] = 1
Input value for list2[5] = 4
Input value for list2[6] = 3
Input value for list2[7] = 5
Input value for list2[8] = 2
Input value for list2[9] = 1
```

```
List1 : 1  3  4  2  1  5  4  2  4  2
List2 : 5  3  2  5  1  4  3  5  2  1
List3 : 6  6  6  7  2  9  7  7  6  3
```

3. Revise your **`List2.java`** program. Thereafter, your program should display the highest value in **list3** and the lowest value in **list3**.
   3.1. Write the additional codes here:

```
int mataas = list3[0];
int mababa = list3[0];

for (int i = 1; i < 10; i++) {
    if (list3[i] > mataas) {
        mataas = list3[i];
    }
    if (list3[i] < mababa) {
        mababa = list3[i];
    }
}

System.out.println("Highest value in List3: " + mataas);
System.out.println("Lowest value in List3: " + mababa);
```

4. Revise your **`List2.java`** program. Thereafter, your program should allow the user to input an integer value to be searched in **list3**. Your program should display whether the inputted integer value is found in **list3**, how many of it is in **list3**, and what are their locations in **list3**?
   Example output:

```
List1 : 1  3  2  5  7  8  5  6  9  4
List2 : 2  1  4  3  2  1  4  2  0  2
List3 : 3  4  6  8  9  9  9  8  9  6

Input value to search in List3: 9

The value 9 is in List3!
There are 4 of it in List3.
Located at: list3[4], list3[5], list3[6], list3[8]
```

   4.1. Save then compile your program until no errors and warnings are reported.
   4.2. Run your program.

```
Enter Ten(10) Integer Values for List1
Input value for list1[0] = 1
Input value for list1[1] = 4
Input value for list1[2] = 5
Input value for list1[3] = 3
Input value for list1[4] = 2
Input value for list1[5] = 1
Input value for list1[6] = 5
Input value for list1[7] = 4
Input value for list1[8] = 5
Input value for list1[9] = 3
```

4.3. Simulate and write what will be displayed on the screen.

```
Enter Ten(10) Integer Values for List2
Input value for list2[0] = 1
Input value for list2[1] = 4
Input value for list2[2] = 2
Input value for list2[3] = 5
Input value for list2[4] = 4
Input value for list2[5] = 2
Input value for list2[6] = 1
Input value for list2[7] = 4
Input value for list2[8] = 3
Input value for list2[9] = 4

List1 : 1  4  5  3  2  1  5  4  5  3
List2 : 1  4  2  5  4  2  1  4  3  4
List3 : 2  8  7  8  6  3  6  8  8  7

Input value to search in List3: 8

The value 8 is in List3!
There are 4 of it in List 3.
Located at: list3[1]. list3[3]. list3[7]. list3[8]
```

4.4. Write your complete source code here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: List2.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class List2 {
    public static void main(String[] args) {
        String search_input, input;
        boolean hanap = false;
        int search_value, ctr = 0;

        int[] list1 = new int[10];
        int[] list2 = new int[10];
        int[] list3 = new int[10];

        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        try {
            System.out.println("Enter Ten(10) Integer Values for
List1");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list1[" + i + "] =
");
                input = basa.readLine();
                list1[i] = Integer.parseInt(input);
            }
```

```
            System.out.println(" ");
            System.out.println("Enter Ten(10) Integer Values for
List2");
            for (int i = 0; i < 10; i++) {
                System.out.print("Input value for list2[" + i + "] =
");
                input = basa.readLine();
                list2[i] = Integer.parseInt(input);
            }

            System.out.println(" ");
            System.out.print("List1 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list1[i] + "  ");
            }
            System.out.println();

            System.out.print("List2 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list2[i] + "  ");
            }
            System.out.println();

            for (int i = 0; i < 10; i++) {
                list3[i] = list1[i] + list2[i];
            }

            System.out.print("List3 : ");
            for (int i = 0; i < 10; i++) {
                System.out.print(list3[i] + "  ");
            }
            System.out.println();

            System.out.println(" ");
            System.out.print("Input value to search in List3: ");
            search_input = basa.readLine();
            search_value = Integer.parseInt(search_input);

            StringBuilder lokasyon = new StringBuilder();

            System.out.println(" ");
            for (int i = 0; i < 10; i++) {
                if (list3[i] == search_value) {
                    if (!hanap) {
                        System.out.println("The value " + search_value
+ " is in List3!");
                        hanap = true;
                    }
                    ctr++;
                    lokasyon.append("list3[").append(i).append("]. ");
                }
            }
```

```
            if (hanap) {
                System.out.println("There are " + ctr + " of it in List
3.");
                System.out.println("Located at: " +
lokasyon.substring(0, lokasyon.length() - 2));
            } else {
                System.out.println("The value " + search_value + " is
not found in List3");
            }

        } catch (IOException e) {
            System.out.println("Error!");
        }
    }
}
```

5. Create a new program and save it as **`BubbleSort.java`**. Thereafter, the user will be asked to input the size of the list, input values to the list, sort the list in ascending order, and display the sorted list in a horizontalmanner.

    5.1. Write your complete source code here:

```
/* Programmed by: Abenes, Enrico O.
   Program Title: BubbleSort.java
   Program Date: July 11, 2023*/

package intl.cc1;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class BubbleSort {
    public static void main(String args[]) {
        BufferedReader basa = new BufferedReader(new
InputStreamReader(System.in));

        try {
            System.out.print("Enter the size of list: ");
            int laki = Integer.parseInt(basa.readLine());

            int[] list = new int[laki];

            System.out.println(" ");
            System.out.println("Enter the values for the list: ");
            for (int i = 0; i < laki; i++) {
                System.out.print("Value " + (i+1) + ": ");
                list[i] = Integer.parseInt(basa.readLine());
            }
```

```
            for (int i = 0; i < laki - 1; i++) {
                for (int j = 0; j < laki - i - 1; j++) {
                    if (list[j] > list[j + 1]) {
                        int temp = list[j];
                        list[j]= list[j + 1];
                        list[j + 1] = temp;
                    }
                }
            }

            System.out.println(" ");
            System.out.print("Sorted list: ");
            for (int i = 0; i < laki; i++) {
                System.out.print(list[i] + " ");
            }
        } catch (IOException e) {
            System.out.println("Error!");
        } catch (NumberFormatException e) {
            System.out.println("Invalid. Please input a number.");
        }
    }
}
```

1.1. Save then compile your program until no errors and warnings are reported. Run your program.

1.2. Simulate and write what will be displayed on the screen.

```
Enter the size of list: 13

Enter the values for the list:
Value 1: 23
Value 2: 13
Value 3: 53
Value 4: 32
Value 5: 14
Value 6: 24
Value 7: 42
Value 8: 54
Value 9: 23
Value 10: 12
Value 11: 42
Value 12: 54
Value 13: 17

Sorted list: 12 13 14 17 23 23 24 32 42 42 53 54 54
```

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**

Finals Laboratory Activity 4

Name: Abenes, Enrico O.

Subject Code & Schedule: CC1, 1: 30 pm – 5: 20 pm MT

Course and Section: BSIT, INTL1

TITLE: Java Methods

LEARNING OBJECTIVES:

At the end of this activity, the students should be able to:

1. Identify the types of methods in the Java programming language.

2. Identify what portions of the main program can be subdivided into sub-programs.

3. Create methods or sub-programs that can be called in the main program.

4. Distinguish the different parameters used in creating methods.

5. Create a complete Java program that utilizes methods.

INSTRUCTIONS:

1. Make sure you have your own individual account.

2. Always keep your account secret from others to avoid unauthorized access to your files.

3. Always save your work and log off when not using the computer.

4. By now you should have been familiarized with using your text editor.

5. By now you should know how to create, save, compile, execute, and debug programs in Java.

DURATION: Two to Three Meetings

HANDS-ON:

1. Log on using your own individual account. Use your own username and password.

2. Open your text editor.

3. Write your next Java program:

    3.1. Write your next program by copying the source code shown below to your text editor.

```
/* Programmed by: <write your name here>
   Program title: PrintHello.java
   Program Date: <write the date today here> */

public class PrintHello{
     public static void printHello(){
          System.out.print("HELLO");
     }
     public static void printSpace(){
          System.out.print(" ");
     }
     public static void printWorld(){
          System.out.println("WORLD!");
     }
     public static void allTogetherNow(){
          printHello();
          printSpace();
          printWorld();
     }

     public static void main(String[] args){
          AllTogetherNow();
     }
}
```

3.2. Save the program as PrintHello.java then compile your program until no errors and warnings are reported.

3.3. Run your program.

3.4. Simulate and write what will be displayed on the screen.

```
Hello WORLD!
```

3.5. Enumerate all user-defined methods:

printHello() – "Hello" is printed to the console using this way.

printSpace() – This technique prints the character "space" to the console.

printWorld() – "WORLD!" is printed to the console by this way.

allTogetherNow() – The printHello(), printSpace(), and printWorld() methods are sequentially invoked by this method.

3.6. Enumerate all pre-defined methods:

System,out print() – Text can be printed on the console using this technique.

String[] args – This is the input for the main method, a built-in method that acts as the starting point for Java programs.

> ***HINT:*** To call a method, indicate the name of the method preceded by the **()** symbol. User-define methods are methods you created , while pre-defined functions are functions already inherent to the compiler.

4. Is the method parameter always void? No

```
public class OneParam{
     public static void oneParam(int x){
          System.out.println("Number is " + x);
     }

     public static void main(String[] args){
          int x;
          x = 100;
          oneParam(x);
     }
}
```

4.1 To answer that, let's simulate the following program

4.2 Save your program as OneParam.java

4.3 Run and write what will be displayed on the screen?

```
Number is 100
```

4.4. Insert the statement oneParam(75);

What can you conclude? That is, is it possible to use a constant value as a parameter when a method is called?

When you wish to send a method a certain value that is known at compile time and doesn't need to be dynamically computed during runtime, using constant values as method parameters **might be helpful.**

4.5 Declare another integer variable in your main method as follows:

int y; Initialize the variable y to 150. Thereafter, write another statement OneParam(y);

What can you conclude? That is, is it possible to use a variable name that is not the same as that used in the method definition?

Using a variable name other than the one given in the method description is technically **conceivable**. As long as the types of the arguments and parameters match, various variable names can be used to pass values when invoking a method.

4.6 Insert the statement OneParam(x+y);

What can you conclude? That is, is it possible to use an expression (that will evaluate an integer value) as a parameter when the method is called?

When invoking a method, it is **possible** to pass an expression that evaluates to an integer value as a parameter.

4.7 Insert the statement OneParam();

What can you conclude? That is, can you call a method requiring a parameter with a passing one? What error is reported if any?

A method that requires a parameter **cannot be called** without one being sent in. There is a **compilation problem** when OneParam() is called without any parameters.

5. Let us create another program and save it as TwoParams.java

5.1. Copy the source code

```
public class TwoParams{
  public static void twoParams(int x, int y){
     System.out.println("First parameter is " + x);
     System.out.println("Second parameter is " + y);
  }

  public static void main(String[] args){
     int x = 5;
     int y = 10;

     int a;
     int b;

     twoParams(10, 20);
     twoparams(x + 2, y * 10);

     a = -2;
     b = 22;
     twoParams(a, b);
  }
}
```

5.2. Save and then compile your program until no errors and warnings are reported.

5.3. Run your program.

5.4. Simulate and write what will be displayed on the screen.

```
First parameter is 10
Second parameter is 20
First parameter is 7
Second parameter is 100
First parameter is -2
Second parameter is 22
```

5.5. Insert the statement twoParams(y, x);

What can you conclude? Is the result the same as of twoParams(x,y)?

Both "First parameter is 10" and "Second parameter is 5" will be printed. TwoParams(x, y); prints "First parameter is 5" and "Second parameter is 10", however this outcome **is different.**

5.5.1 What will happen if we forgot to type the comma between parameter? For example, will twoParams(x y) work? Java will encounter a **syntax error** as a result. The code won't successfully compile.

5.6. What will happen if we only supplied one parameter? For example, will twoParams(x) work? No, compilation error.

5.7. What will happen if we forgot all the parameters?

For example, will twoParams() work? No, compilation error.

5.8. What will happen if we used more than two parameters? For example, will twoParams(x, y, a) work? No, compilation error.

6. Are all methods of return type void? No

6.1. To answer that let's simulate the following program:

```
//Return value

public class WithReturnValue{
  public static int printSum(int x, int y){
     z = x + y;
     return z;
  }

  public static void main(String[] args){
     int a, b, c;

     System.out.println("Sum = " + printSum(5, 10));
     c = printSum(100, 300);
     System.out.println("c = " + c);

     a = 25;
     b = 75;

     System.out.println("Sum = " + printSum(a, b));
     System.out.println("Sum = " + Sum(a+2, b - 3));

  }
}
```

6.2. Save your program as WithReturnValue.java then compile and run and simulate your program. Write what will be displayed on the screen:

```
Compilation Error

Command execution failed.

BUILD FAILURE
```

6.3. What will happen if the statement return z; was omitted?

When the code attempts to delete the method's return value, a **compilation error** will still result.

6.4. Write a method named printDifference that accepts two integer parameters named x and y then computes the difference (x – y). Call this method inside the main().

```
/* Programmed by: Abenes, Enrico O.
   Program Title: Difference.java
   Program Date: July 27, 2023*/

package intl.cc1;

public class Difference {
    public static int printDifference(int x, int y) {
        int negatibo = x - y;
        return negatibo;
    }

    public static void main(String[] args) {
        int a = 10;
        int b = 5;

        int resulta = printDifference(a, b);
        System.out.println("Difference = " + resulta);
    }
}
```

6.5. Write a method named printProduct that accepts two double-type parameters named x and y. The method should compute and return the value of the product of x and y. Use double as return type. Call this method inside main();

```
/* Programmed by: Abenes, Enrico O.
   Program Title: Product.java
   Program Date: July 27, 2023*/

package intl.cc1;

public class Product {
    public static double printProduct(double x, double y) {
        double producto = x * y;
        return producto;
    }

    public static void main(String[] args) {
        double a = 4.6;
        double b = 5.9;

        double resulta = printProduct(a, b);
        System.out.println("Product = " + resulta);
    }
}
```

Rubrics:

Code Content	The source code submitted covers all the items specified in the activity and satisfactorily meets all these requirements. It also makes use of the specific features of Java specified in the activity.	20 pts
Code Function	The source code submitted works completely with no errors and provides the correct expected output when run.	20 pts
Code Syntax	The source code submitted has sound logic and follows proper syntax for Java, with no unnecessarily complicated code.	10 pts
Total		**100 pts**