Lab 02

Objectives:

1. Taking Input in Java
2. Control Structures
3. Arrays in JAVA

1: Taking input in Java

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.

Step 01: **Import java.util.Scanner;**

Step 02: **Create object of Scanner class**

```
Scanner obj= new Scanner(System.in);
```

Step 03: **Use methods as per data types.**

```
Obj.nextInt(); obj.nextFloat(); obj.nextLong();
```

Similarly we have: nextByte(), nextDouble(), nextBoolean()

For Reading String:

1. next()
2. nextLine();

For Reading a single char:

```
next().charAt(0);
```

Example:

```
import java.util.Scanner;  // Import the Scanner class

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

2: Control Structures

A program is a list of instructions or blocks of instructions. Java provides control structures that can change the path of execution and control the execution of instructions.

There are *three* kinds of control structures in java**:**

i. **Conditional Branches**, which are used for choosing between two or more paths. There are three types in Java: if/else/else if, ternary operator and switch.
ii. **Loops** that are used to iterate through multiple values/objects and repeatedly run specific code blocks. The basic loop types in Java are: for, while and do while.
iii. **Branching Statements**, which are used to alter the flow of control in loops. There are two types in Java: “break” and “continue”.

Conditional Branches:

If / else/ else if:

If a certain condition is true, then if block will be executed otherwise else block will be executed.

Theoretically, we can infinitely chain or nest if/else blocks but this will hurt code readability that's why it's not advised.

Syntax:

```
class ExampleIfElseStatement
{
public static void main(String args[])
{
int age = 15;
if (age > 18) // if age is greater than 18
{
// It will be print if block condition is true.
System.out.println("The age of person is : " + age + "He/she is eligible for
voting");
}
else{

// It will be print if block condition is false.
System.out.println("He/she is not eligible for voting");
}
}
}
```

Ternary Operator:

We can use a ternary operator as a shorthand expression that works like an *if/else* statement.

Syntax:

```
System.out.println(count > 2 ? "Count is higher than 2" : "Count is lower or equal than 2");
```

Switch:

Three or more if/else statements can be hard to read. As one of the possible workarounds, we can use switch, as seen above. If we have multiple cases to choose from, we can use a switch statement.

Syntax:

```
int count = 3;
switch (count) {
case 0:
   System.out.println("Count is equal to 0");
   break;
case 1:
   System.out.println("Count is equal to 1");
   break;
default:
   System.out.println("Count is either negative, or higher than 1");
   break;
}
```

Loops:

Loops are very useful when a programmer wants to execute a statement or block of statement multiple times until certain condition is true.

Syntax:

```
for (int i = 1; i <= 50; i++) {
    System.out.println("Hello World!");
}

int whileCounter = 1;
while (whileCounter <= 50) {
    System.out.println("Hello World!");
    whileCounter++;
}

class ExampleDoWhileLoop
{
public static void main(String args[])
{
int i=1;
do
{
System.out.println(i);
i++;
}while(i <=10);
}
}
```

Branching Statements:

Break:

It is used to exit from a loop or switch statement. A loop would normally go to completion, but *break* would cause the early exit

Syntax:

```
class ExampleBreak
{
public static void main(String args[])
{
for(int i = 1 ; i <= 10 ; i++)
{
System.out.println(i);
if(i == 5)
break;
}
System.out.println("After the for loop");
}
}
```

Continue:

If the condition is true, then the *continue* statement skips the current iteration and transfer the control of the loop immediately to the next iteration

Syntax:

```
public class ContinueExample
{
public static void main(String args[])
{
for (int i = 1; i <= 5; i++)
{
if (i == 2)
{
```

```
continue;
}
System.out.println("Value of i ="+ i);
}
}
}
```

3: Arrays in JAVA

An array stores a sequence of values that are all of the same type. The length of an array is established when the array is created. After creation, its length is fixed. Each item in an array is called an *element*, and each element is accessed by its numerical *index*.

The method that we use to refer to individual values in an array is to number and then *index* them—if we have *n* values, we think of them as being numbered from 0 to *n*−1.

Making an array in a Java program involves three distinct steps:

- Declare the array name.
- Create the array.
- Initialize the array values.

We refer to an array element by putting its index in square brackets after the array name.

To use an array in a program, you must declare a variable to reference the array and specify the array's *element type*.

Syntax:

```
elementType[] arrayRefVar;
```

The **elementType** can be any data type, and all elements in the array will have the same data type. Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array. If a variable does not contain a reference to an array, the value of the variable is **null**. You cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the **new** operator and assign its reference to the variable with the following **syntax**:

```
arrayRefVar = new elementType[arraySize];
```

Java has a shorthand notation, known as the *array initializer*, which combines the declaration, creation, and initialization of an array in one statement using the following **syntax**:

```
elementType[] arrayRefVar = {value0, value1, ..., valuek};
```

Arrays Class

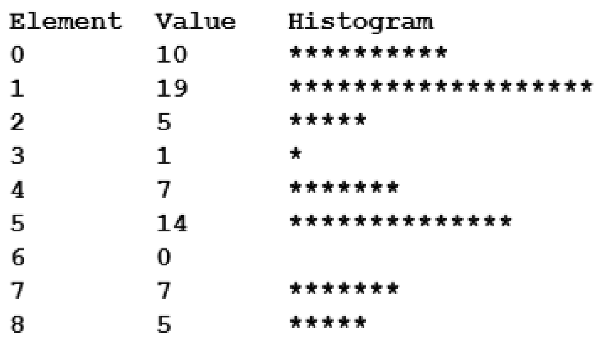
Arrays class which is in java.util.Arrays package, is a provision by Java that provides you a number of methods through which arrays can be manipulated. This class also lets you perform sorting and searching operations on an array.

Lab Tasks:

Exercises

1. Write a program to Find the number of days in a month using a switch statement
2. Write a program to check whether the number is even or odd using switch statement.
3. Write a Java program to copy an array into another array by iterating the first array.
4. Write code that creates an array named odds and stores all odd numbers between 1 and 30 into it using a for loop.
5. Write a Java program to reverse the elements of an array.
6. Write a Java program that takes a jagged array as input and finds the maximum element of array and Display it.
7. Try to implement any sorting algorithm in java like bubble and selection sort.
8. Given the following array, display its data graphically by plotting each numeric value as a bar of asterisks (*) as shown in the diagram.

   int[] array = {10, 19, 5, 1, 7, 14, 0, 7, 5};



```
Element  Value   Histogram
0        10      **********
1        19      *******************
2        5       *****
3        1       *
4        7       *******
5        14      **************
6        0
7        7       *******
8        5       *****
```