Experiment: Java PATH Setup

Aim: To install Java Development Kit (JDK) and configure JAVA_HOME and PATH environment variables.

Theory: Java programs require the Java Development Kit (JDK) for compilation and execution. Setting the PATH variable allows Java commands such as java and javac to be executed from any directory.

Requirements:
• Operating System: Windows / Linux / macOS
• Java Development Kit (JDK)

Procedure:

Step 1: JDK Installation
1. Download the Java Development Kit from the official Java website.
2. Install the JDK using default settings.
3. Note the installation path (Example: C:\Program Files\Java\jdk-17).

Step 2: Setting JAVA_HOME (Windows)
1. Right-click on this PC and select Properties.
2. Click on Advanced system settings.
3. Click on Environment Variables.
4. Under System Variables, click New.
5. Set Variable Name as JAVA_HOME.
6. Set Variable Value as the JDK installation path.
7. Click OK.

Step 3: Setting PATH Variable

1. Under System Variables, select Path and click Edit.
2. Click New and add %JAVA_HOME%\bin.
3. Click OK to save changes.

Step 4: Verification
1. Open Command Prompt.
2. Execute the commands:

```
java -version
javac -version
```

Output:
The installed Java version is displayed on the command prompt.

Result:
Thus, Java PATH was successfully configured and verified.

Experiment 1: Write a basic java program to implement basic programming concepts such as

(a) Input Handling in Java

Aim: To write a Java program to demonstrate input handling using the Scanner class.

Theory: The Scanner class in Java is used to take input from the user during runtime.

Algorithm:
1. Import Scanner class.
2. Create a Scanner object.
3. Accept integer input.
4. Display the input.

Program Code:

```
import java.util.Scanner;

public class InputHandling {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter an integer: ");
      int num = sc.nextInt();
      System.out.println("You entered: " + num);
      sc.close();
   }
}
```

Result:The program successfully accepts and displays user input.

(b): Arithmetic Operations in Java

Aim: To write a Java program to perform basic arithmetic operations.

Theory: Java supports arithmetic operators such as +, -, *, and /.

Algorithm:
1. Declare two variables.
2. Perform arithmetic operations.
3. Display the results.

Program Code:

```
public class ArithmeticOperations {
   public static void main(String[] args) {
      int a = 20;
      int b = 10;
      System.out.println("Addition = " + (a + b));
      System.out.println("Subtraction = " + (a - b));
      System.out.println("Multiplication = " + (a * b));
      System.out.println("Division = " + (a / b));
   }
}
```

Result:
The program performs arithmetic operations correctly.

(c): Control Flow Structures in Java

Aim: To write a Java program to demonstrate control flow structures.

Theory: Control flow statements determine the execution order.

Algorithm:
1. Declare a variable.
2. Use if–else for condition.
3. Use for loop.

Program Code:

```
public class ControlFlowDemo {
   public static void main(String[] args) {
      int number = 7;
      if (number % 2 == 0)
         System.out.println("Even number");
      else
         System.out.println("Odd number");

      for (int i = 1; i <= 5; i++)
         System.out.println(i);
   }
}
```

Result:
The program demonstrates control flow successfully.

(d): Data Storage Using Arrays

Aim: To write a Java program to demonstrate data storage using arrays.

Theory: Arrays store multiple values of the same data type.

Algorithm:
1. Declare and initialize array.
2. Traverse the array.
3. Display elements.

Program Code:

```
public class ArrayDemo {
   public static void main(String[] args) {
      int[] arr = {10, 20, 30, 40, 50};
      for (int i = 0; i < arr.length; i++)
         System.out.println(arr[i]);
   }
}
```

Result:
The program stores and displays array elements successfully.

Method	Description
`nextLine()`	Reads a `string` value from the user
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user