**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG**

**Department of Computer Science and Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025-2026** | **Estimated Time** | **Experiment No. 1 -Batch C** |
| **Course & Semester** | **S.E. CSE** | **Subject Name** | **Object Oriented**  **Programming with Java Lab** |
| **Module No.** | **01** | **Chapter Title** |  |
| **Experiment Type** | **Software Performance** | **Subject Code** | 25PCC12CS07 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of Student** | Aaradhya Raut | **Roll No.** | 10877 |
| **Date of Performance.:** | 24/08/2025 | **Date of Submission.:** | 08/08/2025 |
| **CO Mapping** | **CO1. Implement Core Java Concepts.** | | |

**Objective of Experiment:**

Study of Understanding variables and data types, Basic input/output operations, Control Structures and Functions in Java.

**Pre-Requisite:** Any programming language like C, C++

**Tools:** Java IDLE

**Theory:**

**A. Data Types:** In Java, data types specify the size and type of values that can be stored in a variable. Java is a strongly typed language, which means each variable must be declared with a data type.

**Primitive Data Types:**Java has eight built-in primitive data types:

* **byte**: 8-bit integer. Range: -128 to 127. Example: byte b = 10;
* **short**: 16-bit integer. Range: -32,768 to 32,767. Example: short s = 1000;
* **int**: 32-bit integer. Range: -2^31 to 2^31-1. Example: int i = 12345;
* **long** :Size: 64-bit, Range: -2^63 to 2^63-1, Example: long l = 123456789L;
* **float**: Size: 32-bit (single-precision).Example: float f = 12.34f;
* **double**: Size: 64-bit (double-precision).Example: double d = 123.456;
* **Boolean**: Size: not precisely defined, typically a bit or a byte. Values: true or false . Example: boolean isJavaFun = true;
* **Char**: Size: 16-bit (Unicode character). Range: '\u0000' to '\uffff'. Example: char c = 'A';

**Non-Primitive Data Types (Reference Types):** These are more complex types that are derived from the primitive data types:

* **Strings**: Example: String str = "Hello, Java!";
* **Arrays**: Example: int[] arr = {1, 2, 3, 4, 5};
* **Classes**: Example: MyClass obj = new MyClass();
* **Interfaces**: Example: Runnable r = new MyRunnable();

**B**. **Variables :** Variables are containers for storing data values. In Java, variables must be declared before they can be used. The declaration involves specifying the variable's type and name.

### Variable Naming Conventions

* Variable names should be meaningful and describe the purpose of the variable.
* The first character must be a letter, a dollar sign ($), or an underscore (\_).
* Subsequent characters can be letters, digits, dollar signs, or underscores.
* Variable names are case-sensitive.
* Avoid using reserved keywords as variable names.

#### Types of Variables

|  |  |
| --- | --- |
| **Description** | **Example** |
| **Local Variables**  Declared inside a method or a block.  Scope is limited to the block or method where it is declared.  Must be initialized before use. | void myMethod()  { int x = 10; // local variable System.out.println(x); } |
| **Instance Variables (Non-static Fields)**  Declared inside a class but outside any method, constructor, or block.  Each instance of the class has its own copy of the variable.  Initialized to default values if not explicitly initialized | public class MyClass {  int instanceVariable; // instance variable  MyClass(int value) {  instanceVariable = value;  }  } |
| **Class Variables (Static Fields)**  Declared with the static keyword inside a class but outside any method, constructor, or block.  Shared among all instances of the class.  Initialized to default values if not explicitly initialized. | public class MyClass {  static int staticVariable; // class variable  } |

### C. Input and Output in Java :Java provides various classes and methods for handling input and output operations. The most commonly used classes are part of the java.io package.

|  |  |
| --- | --- |
| Output in Java | |
| **Using System.out.print and System.out.println**   * System.out.print: Prints text to the console without a newline at the end. * System.out.println: Prints text to the console with a newline at the end | System.out.print("Hello, ");System.out.println("World!"); |
| Using System.out.printfAllows formatted output, similar to printf in C | System.out.printf("Name: %s, Age: %d\n", "Alice", 30); |
| **Input in Java** | |
| **Reading from the Console** : **Using Scanner**  Scanner is a simple text scanner which can parse primitive types and strings using regular expressions.  [ Scanner class is a part of the **java.util** package ]  **Reading Strings:**  next(): Reads the next token as a string (delimited by whitespace).  nextLine(): Reads the entire next line of text.  **Reading Primitive Data Types:**  nextInt(): Reads the next token as an integer.  nextDouble(): Reads the next token as a double.  nextFloat(): Reads the next token as a float.  nextLong(): Reads the next token as a long.  nextBoolean(): Reads the next token as a boolean | import java.util.Scanner;  public class Main {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter your name: ");  String name = scanner.nextLine();  System.out.print("Enter your age: ");  int age = scanner.nextInt();  System.out.println("Name: " + name + ", Age: " + age);  scanner.close();  }  } |

D. Control Structures: Control structures in Java allow you to control the flow of execution in your programs. They are divided into three main categories: decision-making structures, looping structures, and branching structures.

|  |  |  |
| --- | --- | --- |
| **Decision-Making Structures** | | |
| **if Statement**  Executes a block of code if a specified condition is true | if (condition) {  // code to be executed if condition is true  } | if (num > 0)  { System.out.println("The number is positive."); } |
| **if-else Statement**  Executes one block of code if a condition is true, and another block if it is false | if (condition) {  // code to be executed if condition is true  } else {  // code to be executed if condition is false  } | if (num > 0)  { System.out.println("The number is positive."); }  else { System.out.println("The number is not positive.");  } |
| **if-else if-else Statement**  Tests multiple conditions sequentially. | if (condition1) {  // code to be executed if condition1 is true  } else if (condition2) {  // code to be executed if condition2 is true  } else {  // code to be executed if both conditions are false  } | if (age < 0) {  System.out.println("Invalid age.");  } else if (age >= 0 && age <= 12) {  System.out.println("Child.");  } else if (age >= 13 && age <= 19) {  System.out.println("Teenager.");  } else if (age >= 20 && age <= 64) {  System.out.println("Adult.");  } else if (age >= 65) {  System.out.println("Senior.");  } else {  System.out.println("Invalid age range.");  } |
| **switch Statement**  Selects one of many blocks of code to execute based on the value of an expression. | switch (expression) {  case value1:  // code to be executed if expression == value1  break;  case value2:  // code to be executed if expression == value2  break;  // you can have any number of case statements  default:  // code to be executed if none of the cases match  } | switch (num) {  case 1: System.out.println("The number is one.");  break;  case 10: System.out.println("The number is ten.");  break;  default: System.out.println("The number is neither one nor ten.");  } |
| **Looping Structures** | | |
| **while Loop**  Repeatedly executes a block of code as long as a specified condition is true. | while (condition) {  // code to be executed  } | int i = 0;  while (i < 5) {  System.out.println("While loop: " + i);  i++;  } |
| **do-while Loop**  Similar to the while loop, but the block of code is executed at least once before the condition is tested. | do {  // code to be executed  } while (condition); | int j = 0;  do {  System.out.println("Do-while loop: " + j);  j++;  } while (j < 5); |
| **for Loop**  Executes a block of code a specific number of times. | for (initialization; condition; increment/decrement) {  // code to be executed  } | for (int k = 0; k < 5; k++) { System.out.println("For loop: " + k);  } |
| **Enhanced for Loop (for-each)**  Used to iterate over arrays or collections | for (type variable : array/collection) {  // code to be executed  } | int[] numbers = {1, 2, 3, 4, 5};  for (int num : numbers) { System.out.println("Enhanced for loop: " + num);  } |
| **Branching Structures** | | |
| **break Statement**  Exits a loop or a switch statement immediately. | for (int i = 0; i < 10; i++) {  if (i == 5) {  break; // exit the loop when i is 5  }  System.out.println(i);  } | for (int i = 0; i < 10; i++) {  if (i == 5) {  break; // exit the loop when i is 5  }  System.out.println("Break statement: " + i); } |
| **continue Statement**  Skips the current iteration of a loop and proceeds with the next iteration. | for (int i = 0; i < 10; i++) {  if (i == 5) {  continue; // skip the current iteration when i is 5  }  System.out.println(i);  } | for (int i = 0; i < 10; i++) {  if (i == 5) {  continue; // skip the current iteration when i is 5 }  System.out.println("Continue statement: " + i); } |
| **return Statement**  Exits from the current method and optionally returns a value | public int sum(int a, int b) {  return a + b; // exit the method and return the sum  } | public int add(int a, int b) {  return a + b; // exit the method and return the sum  } |

**E. Arrays:** An array in Java is a collection of elements, all of the same type, stored in a contiguous memory location. Arrays are used to store multiple values in a single variable, making it easier to manage large amounts of data.

|  |  |
| --- | --- |
| **Single Dimensional Array** | |
| **Declaration:**  To declare an array, specify the data type of its elements followed by square brackets [] and the array name. | int[] numbers; |
| **Initialization:**  Arrays can be initialized when they are declared or later in the code. | // Declaration and memory allocation  numbers = new int[5];  // Declaration and initialization together  int[] numbers = new int[5];  // Initialization with values  int[] numbers = {1, 2, 3, 4, 5}; |
| **Accessing Array Elements**  Array elements are accessed using their index, starting from 0. | int firstNumber = numbers[0]; // Access the first element  numbers[2] = 10; // Modify the third element |
| **Length of an Array**  The length of an array can be determined using the length property. | int length = numbers.length; |
| **Iterating Over an Array**  You can use loops to iterate over the elements of an array. | for (int i = 0; i < numbers.length; i++) {  System.out.println(numbers[i]);  }  Example 2:  for (int number : numbers) {  System.out.println(number);  } |
| **2D Array** | |
| **Declaration:** | int[][] matrix; |
| **Initialization:** | // Declaration and memory allocation  matrix = **new** int[3][3];  // Declaration and initialization together  int[][] matrix = {  {1, 2, 3},  {4, 5, 6},  {7, 8, 9}  }; |
| **Accessing Elements:** | int value = matrix[1][2]; // Access element in second row, third column  matrix[0][0] = 10; // Modify element in first row, first column |
| **Iterating Over Multi-Dimensional Arrays:** | for (int i = 0; i < matrix.length; i++) {  for (int j = 0; j < matrix[i].length; j++) {  System.out.println(matrix[i][j]);  }  } |

**Problem Description:**

|  |  |
| --- | --- |
| **Stock Market Problem:** Develop a program for the Stock Market that calculates the total price for the customer's stocks, including interest values and total time duration, and prints the Invoice. | |
| Input:  **List of current stocks and Prices**: A list of available stocks with their prices.  **Customer stocks**: The specific stocks selected by the customer.  **Interest Information**: % of interest. | Expected Output:   * List of stocks selected with their prices * To interest calculated for duration * Total stock price * Invoice |

**Post Lab Questions:**

|  |
| --- |
|  |
| 1. Write a Java program that takes a percentage score as input from the user and prints the corresponding grade (A, B, C, D, F). Use a function to determine the grade. [ Use an if-else if-else ladder to determine the grade.] Below 40-Grade F ,Between 41 to 60- Grade D ,Between 61-80- Grade C, Between 81-90- Grade B, Above 91- Grade A |
| 1. Write a Java program that simulates a simple banking system. The program should allow the user to deposit money, withdraw money, and check the balance. Use functions to handle each operation.   Hints: Use the Scanner class for input. Create functions void deposit(double amount), void withdraw(double amount), and double checkBalance(). Use a switch statement in the main method to handle user choices. |

|  |  |  |  |
| --- | --- | --- | --- |
| **On time Completion and Submission (2)** | **Knowledge of the topic (4)** | **Implementation and Output (4)** | **Total (10)** |
|  |  |  |  |

**References:**

|  |  |
| --- | --- |
| **Study Materials**  [**https://www.w3schools.com/java/**](https://www.w3schools.com/java/)  [**https://www.geeksforgeeks.org/java/**](https://www.geeksforgeeks.org/java/)  https://www.codecademy.com/learn/learn-java | **Video Channels**:  [**https://www.youtube.com/user/programmingwithmosh**](https://www.youtube.com/user/programmingwithmosh)  [**https://www.youtube.com/c/TheNetNinja**](https://www.youtube.com/c/TheNetNinja)  [**https://www.youtube.com/c/Freecodecamp**](https://www.youtube.com/c/Freecodecamp)  [**https://www.youtube.com/user/Simplilearn**](https://www.youtube.com/user/Simplilearn) |
| **Study Materials used for Demo** | |

**Note:-students are expected to paste screenshot of the program with output**

Problem Statement Output:

Post Lab Q.1 Output:

Post Lab Q.2 Output: