**FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG**

**Department of Computer Science and Engineering**

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2025-2026** | **Estimated Time** | **Experiment No. 1** |
| **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** | Atharva Dharmendra Jagtap | **Roll No.** |  |
| **Date of Performance.:** |  | **Date of Submission.:** |  |
| **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:**

|  |  |
| --- | --- |
| **Coffee Shop Problem:** Develop a program for a coffee shop that calculates the total cost of a customer's order, including taxes and discounts, and prints the receipt. | |
| Input:  **Menu Items and Prices**: A list of available coffee and other items with their prices.  **Customer Order**: The specific items ordered by the customer and their quantities.  **Discount Information**: Any applicable discounts (e.g., percentage discount, loyalty discounts).  **Tax Rate**: The tax rate to be applied to the total order. | Expected Output:   * List of items ordered with their quantities and prices * Subtotal before tax and discounts * Total discount applied * Tax amount * Final total amount to be paid |

**Code:**

// MenuItem.java

public class MenuItem {

    private String name;

    private double price;

    public MenuItem(String name, double price) {

        this.name = name;

        this.price = price;

    }

    public String getName() {

        return name;

    }

    public double getPrice() {

        return price;

    }

    // Optional: for easy printing of menu items

    @Override

    public String toString() {

        return name + " - $" + String.format("%.2f", price);

    }

}

// OrderItem.java

public class OrderItem {

    private MenuItem item;

    private int quantity;

    public OrderItem(MenuItem item, int quantity) {

        this.item = item;

        this.quantity = quantity;

    }

    public MenuItem getItem() {

        return item;

    }

    public int getQuantity() {

        return quantity;

    }

    // Calculate the total cost for this specific order item (item price \* quantity)

    public double getTotalCost() {

        return item.getPrice() \* quantity;

    }

    // Optional: for easy printing of order items

    @Override

    public String toString() {

        return String.format("%-25s x %d = $%.2f", item.getName(), quantity, getTotalCost());

    }

}

// CoffeeShop.java

import java.util.Scanner;

import java.util.ArrayList; // Using ArrayList for dynamic order items

import java.util.Arrays;    // For fixed menu array

public class CoffeeShop {

    // Fixed Menu Items (using an array)

    private MenuItem[] menu;

    private double taxRate;

    private double discountPercentage; // Global discount for simplicity

    public CoffeeShop(double taxRate, double discountPercentage) {

        this.taxRate = taxRate;

        this.discountPercentage = discountPercentage;

        initializeMenu();

    }

    private void initializeMenu() {

        menu = new MenuItem[] {

            new MenuItem("Espresso", 2.50),

            new MenuItem("Latte", 4.00),

            new MenuItem("Cappuccino", 3.75),

            new MenuItem("Americano", 3.00),

            new MenuItem("Muffin", 2.75),

            new MenuItem("Croissant", 3.25),

            new MenuItem("Orange Juice", 2.00)

        };

    }

    public void displayMenu() {

        System.out.println("\n--- Our Menu ---");

        for (int i = 0; i < menu.length; i++) {

            System.out.println((i + 1) + ". " + menu[i].toString());

        }

        System.out.println("----------------\n");

    }

    // Helper to find a MenuItem by its name (case-insensitive)

    public MenuItem findMenuItem(String itemName) {

        for (MenuItem item : menu) {

            if (item.getName().equalsIgnoreCase(itemName)) {

                return item;

            }

        }

        return null; // Not found

    }

    // Helper to find a MenuItem by its index (from displayed menu)

    public MenuItem getMenuItemByIndex(int index) {

        if (index >= 0 && index < menu.length) {

            return menu[index];

        }

        return null;

    }

    public void processOrder() {

        Scanner scanner = new Scanner(System.in);

        ArrayList<OrderItem> customerOrder = new ArrayList<>(); // Using ArrayList for order, more flexible than fixed array

        double subtotal = 0;

        System.out.println("Welcome to the Coffee Shop!");

        displayMenu();

        System.out.println("Enter your order (type item number and quantity, e.g., '1 2' for 2 Espressos).");

        System.out.println("Type 'done' when finished.");

        while (true) {

            System.out.print("Order item (number quantity or 'done'): ");

            String input = scanner.nextLine().trim();

            if (input.equalsIgnoreCase("done")) {

                break;

            }

            try {

                String[] parts = input.split(" ");

                if (parts.length == 2) {

                    int itemNumber = Integer.parseInt(parts[0]);

                    int quantity = Integer.parseInt(parts[1]);

                    if (quantity <= 0) {

                        System.out.println("Quantity must be positive. Please try again.");

                        continue;

                    }

                    MenuItem selectedItem = getMenuItemByIndex(itemNumber - 1); // -1 because menu is 0-indexed

                    if (selectedItem != null) {

                        OrderItem orderItem = new OrderItem(selectedItem, quantity);

                        customerOrder.add(orderItem);

                        subtotal += orderItem.getTotalCost();

                        System.out.println("Added: " + orderItem.toString());

                    } else {

                        System.out.println("Invalid item number. Please refer to the menu.");

                    }

                } else {

                    System.out.println("Invalid input format. Please enter 'item\_number quantity'.");

                }

            } catch (NumberFormatException e) {

                System.out.println("Invalid number format. Please enter valid numbers for item and quantity.");

            } catch (Exception e) {

                System.out.println("An unexpected error occurred: " + e.getMessage());

            }

        }

        if (customerOrder.isEmpty()) {

            System.out.println("No items ordered. Order cancelled.");

            // scanner.close(); // Close scanner here if no receipt is generated

            return;

        }

        generateReceipt(customerOrder, subtotal);

        scanner.close(); // Close scanner once all input is done

    }

    private void generateReceipt(ArrayList<OrderItem> order, double subtotal) {

        System.out.println("\n\n--- Your Receipt ---");

        System.out.println("Items Ordered:");

        for (OrderItem item : order) {

            System.out.println("  " + item.getItem().getName() + " x " + item.getQuantity() +

                               " @ $" + String.format("%.2f", item.getItem().getPrice()) +

                               " = $" + String.format("%.2f", item.getTotalCost()));

        }

        System.out.println("--------------------");

        System.out.printf("Subtotal: $%.2f%n", subtotal);

        double discountAmount = subtotal \* discountPercentage;

        System.out.printf("Discount (%.0f%%): -$%.2f%n", (discountPercentage \* 100), discountAmount);

        double subtotalAfterDiscount = subtotal - discountAmount;

        System.out.printf("Subtotal after discount: $%.2f%n", subtotalAfterDiscount);

        double taxAmount = subtotalAfterDiscount \* taxRate;

        System.out.printf("Tax (%.1f%%): +$%.2f%n", (taxRate \* 100), taxAmount);

        double finalTotal = subtotalAfterDiscount + taxAmount;

        System.out.printf("Final Total: $%.2f%n", finalTotal);

        System.out.println("--------------------");

        System.out.println("Thank you for your order!");

    }

    public static void main(String[] args) {

        // Initialize CoffeeShop with a 8% tax rate and 10% discount

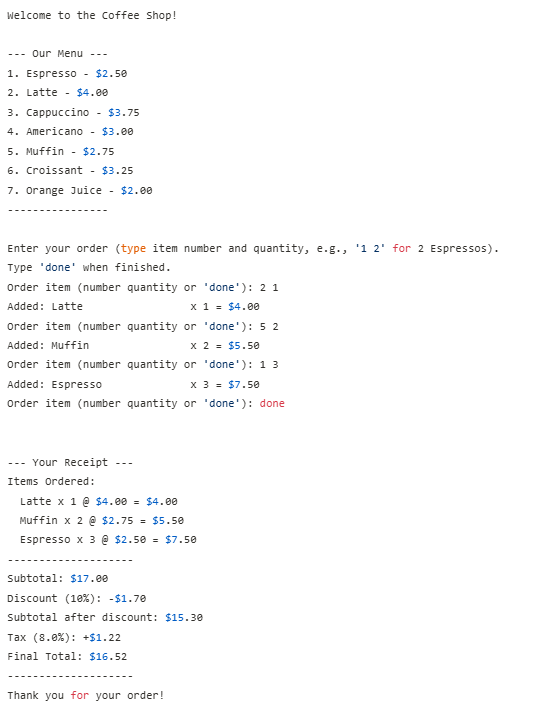
        CoffeeShop myCoffeeShop = new CoffeeShop(0.08, 0.10);

        myCoffeeShop.processOrder();

    }

}

**Output:**

****

**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. |

**Q1. Ans.**

Code:

import java.util.Scanner;

public class GradeCalculator {

    public static char getGrade(double score) {

        if (score < 0 || score > 100) {

            System.out.println("Invalid score: Score must be between 0 and 100.");

            return 'X';

        } else if (score >= 91) {

            return 'A';

        } else if (score >= 81) {

            return 'B';

        } else if (score >= 61) {

            return 'C';

        } else if (score >= 41) {

            return 'D';

        } else {

            return 'F';

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("--- Grade Calculator ---");

        System.out.print("Enter the percentage score: ");

        // Check if the user input is a valid number

        if (scanner.hasNextDouble()) {

            double userScore = scanner.nextDouble();

            char grade = getGrade(userScore);

            if (grade != 'X') { // Only print the grade if it's valid

                System.out.println("The corresponding grade is: " + grade);

            }

        } else {

            System.out.println("Invalid input. Please enter a numeric score.");

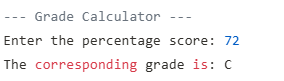
        }

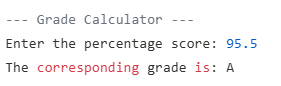
        scanner.close();

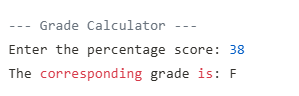
    }

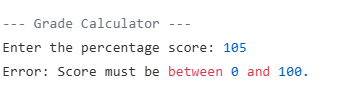
}

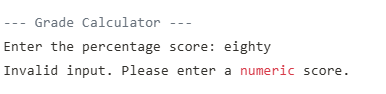
Output:











**Q2. Ans.**

Code:

import java.util.Scanner;

public class BankingSystem {

    public static void deposit(double amount) {

        if (amount > 0) {

            balance += amount; // Add the amount to the balance

            System.out.printf("Successfully deposited $%.2f. New balance: $%.2f%n", amount, balance);

        } else {

            System.out.println("Deposit amount must be positive.");

        }

    }

    public static void withdraw(double amount) {

        if (amount > 0) {

            if (balance >= amount) {

                balance -= amount; // Subtract the amount from the balance

                System.out.printf("Successfully withdrew $%.2f. New balance: $%.2f%n", amount, balance);

            } else {

                System.out.println("Insufficient funds. Current balance: $" + String.format("%.2f", balance));

            }

        } else {

            System.out.println("Withdrawal amount must be positive.");

        }

    }

    public static double checkBalance() {

        return balance;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int choice;

        System.out.println("--- Welcome to Simple Banking System ---");

        do {

            System.out.println("\nSelect an option:");

            System.out.println("1. Deposit");

            System.out.println("2. Withdraw");

            System.out.println("3. Check Balance");

            System.out.println("4. Exit");

            System.out.print("Enter your choice: ");

            // Input validation for choice

            if (scanner.hasNextInt()) {

                choice = scanner.nextInt();

                scanner.nextLine(); // Consume newline left-over

                switch (choice) {

                    case 1:

                        System.out.print("Enter amount to deposit: $");

                        if (scanner.hasNextDouble()) {

                            double depositAmount = scanner.nextDouble();

                            scanner.nextLine(); // Consume newline

                            deposit(depositAmount);

                        } else {

                            System.out.println("Invalid input. Please enter a numeric amount.");

                            scanner.nextLine(); // Consume invalid input

                        }

                        break;

                    case 2:

                        System.out.print("Enter amount to withdraw: $");

                        if (scanner.hasNextDouble()) {

                            double withdrawAmount = scanner.nextDouble();

                            scanner.nextLine(); // Consume newline

                            withdraw(withdrawAmount);

                        } else {

                            System.out.println("Invalid input. Please enter a numeric amount.");

                            scanner.nextLine(); // Consume invalid input

                        }

                        break;

                    case 3:

                        System.out.printf("Current balance: $%.2f%n", checkBalance());

                        break;

                    case 4:

                        System.out.println("Thank you for using Simple Banking System. Goodbye!");

                        break;

                    default:

                        System.out.println("Invalid choice. Please select a number between 1 and 4.");

                }

            } else {

                System.out.println("Invalid input. Please enter a number for your choice.");

                scanner.nextLine(); // Consume invalid input to prevent infinite loop

                choice = 0; // Set choice to 0 to keep the loop running

            }

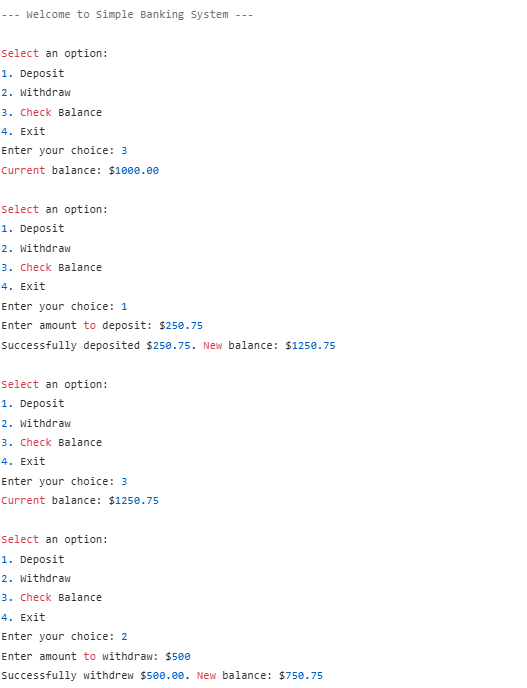
        } while (choice != 4); // Loop until the user chooses to exit

        scanner.close();

    }

}

Output:



|  |  |  |  |
| --- | --- | --- | --- |
| **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**  <Add links here> | |

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