AMRITA VISHWA VIDYAPEETAM

AMRITA SCHOOL OF ENGINEERING, CHENNAI

19CSE204 -OBJECT ORIENTED PARADIGM

SUBMITTED BY:

SUVETHA SP

CH.EN.U4CCE22041

INPUT AND OUTPUT STATEMENTS

OBJECTIVE

To develop a simple Java application that simulates basic banking operations like accepting user details, depositing and withdrawing money, validating transactions, and displaying the final account summary using user input and output statements.

OVERVIEW:

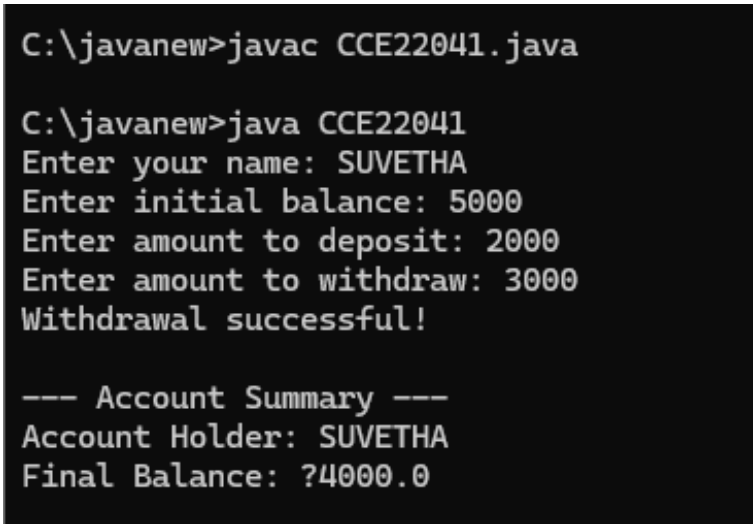
The program accepts user input for name and initial balance, allows money to be deposited and withdrawn, checks for sufficient balance during withdrawal, and finally displays the account summary. The Scanner class is used for input, and System.out.println is used for output.

CODE :

```
import java.util.Scanner;

public class CCE22041 {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);


        // Input: User's name

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

        String name = scanner.nextLine();


        // Input: Initial balance

        System.out.print("Enter initial balance: ");

        double balance = scanner.nextDouble();


        // Input: Amount to deposit

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

        double deposit = scanner.nextDouble();

        balance += deposit;


        // Input: Amount to withdraw
```

```
        System.out.print("Enter amount to withdraw: ");
        double withdraw = scanner.nextDouble();

        // Check if withdrawal is possible
        if (withdraw > balance) {
            System.out.println("Insufficient balance! Withdrawal denied.");
        } else {
            balance -= withdraw;
            System.out.println("Withdrawal successful!");
        }

        // Output: Final account summary
        System.out.println("\n--- Account Summary ---");
        System.out.println("Account Holder: " + name);
        System.out.println("Final Balance: ₹" + balance);

        scanner.close();
    }
}
```

OUTPUT:



DATA OPERATIONS

OBJECTIVE:

To create a Java application that evaluates an employee's salary and performance status using all fundamental data operations including arithmetic, relational, logical, and unary operators.

OVERVIEW

This program collects employee details such as name, basic salary, number of working days, leaves taken, and target completion status. Using these inputs, it:

- Calculates daily salary, deductions, and net salary using **arithmetic operations**.
- Applies **unary operators** to simulate reward point updates.
- Uses **relational operators** to evaluate attendance and performance conditions.
- Combines **logical operations** to determine award eligibility and HR warnings.
- Displays a detailed evaluation report.

CODE :

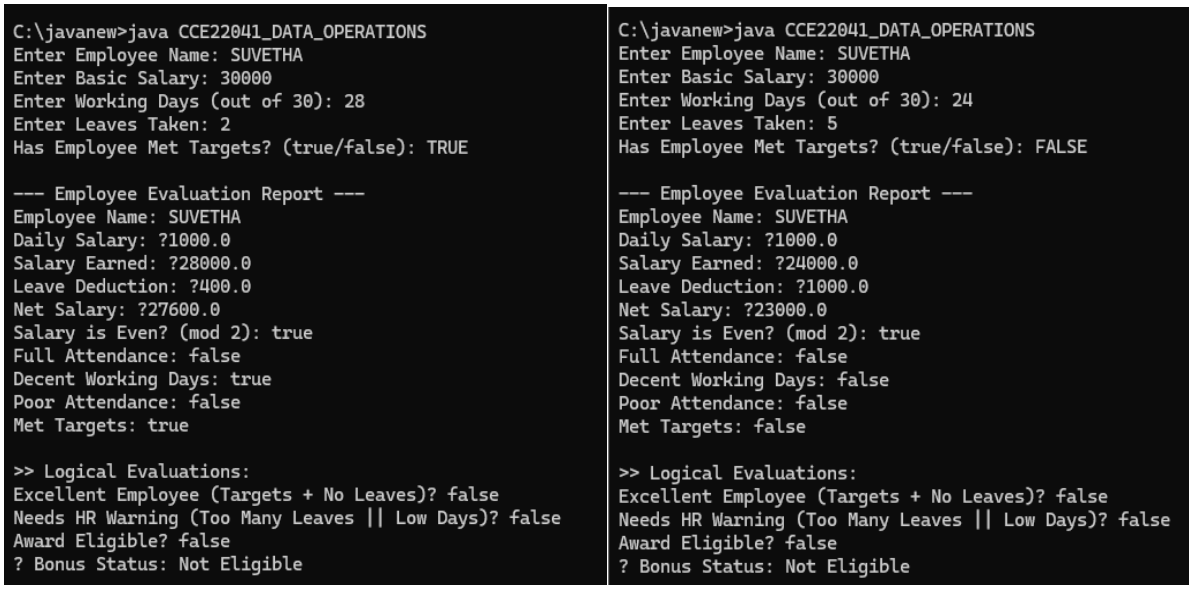
```
import java.util.Scanner;



public class CCE22041_DATA_OPERATIONS {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);



        // Input

        System.out.print("Enter Employee Name: ");

        String name = scanner.nextLine();



        System.out.print("Enter Basic Salary: ");

        double basicSalary = scanner.nextDouble();



        System.out.print("Enter Working Days (out of 30): ");
```

```
int workingDays = scanner.nextInt();

System.out.print("Enter Leaves Taken: ");
int leaves = scanner.nextInt();

System.out.print("Has Employee Met Targets? (true/false): ");
boolean metTargets = scanner.nextBoolean();

// --------- Arithmetic Operators ---------
double dailySalary = basicSalary / 30;              // Division
double salaryEarned = dailySalary * workingDays;    // Multiplication
double leaveDeduction = leaves * 200;               // Subtraction
double netSalary = salaryEarned - leaveDeduction;
double remainder = (int)salaryEarned % 2;           // Modulus

// --------- Unary Operators ---------
int rewardPoints = 0;
rewardPoints++;  // increment
rewardPoints--;  // decrement (back to 0)

// --------- Relational Operators ---------
boolean fullAttendance = (leaves == 0);         // ==
boolean poorAttendance = (leaves > 5);          // >
boolean decentWork = (workingDays >= 25);       // >=
```

```
boolean lowWork = (workingDays <= 15);          // <=
boolean bonusSalary = (netSalary >= 40000);     // >=
boolean bonusEligible = !(leaves != 0);         // != + !

// --------- Logical Operators ---------
boolean excellentEmployee = (metTargets && fullAttendance); // &&
boolean warningRequired = (poorAttendance || lowWork);      // ||
boolean notEligibleForAward = !excellentEmployee;           // !

// --------- Output ---------
System.out.println("\n--- Employee Evaluation Report ---");
System.out.println("Employee Name: " + name);
System.out.println("Daily Salary: ₹" + dailySalary);
System.out.println("Salary Earned: ₹" + salaryEarned);
System.out.println("Leave Deduction: ₹" + leaveDeduction);
System.out.println("Net Salary: ₹" + netSalary);
System.out.println("Salary is Even? (mod 2): " + (remainder == 0));

// Logical Output
System.out.println("Full Attendance: " + fullAttendance);
System.out.println("Decent Working Days: " + decentWork);
System.out.println("Poor Attendance: " + poorAttendance);
System.out.println("Met Targets: " + metTargets);
```

```
        System.out.println("\n>> Logical Evaluations:");

        System.out.println("Excellent Employee (Targets + No Leaves)? " + excellentEmployee);

        System.out.println("Needs HR Warning (Too Many Leaves || Low Days)? " + warningRequired);

        System.out.println("Award Eligible? " + !notEligibleForAward);



        if (bonusSalary && excellentEmployee) {

            System.out.println("☑ Bonus Status: Eligible");

        } else {

            System.out.println("❌ Bonus Status: Not Eligible");

        }



        scanner.close();

    }

}
```

OUTPUT:



```
C:\javanew>java CCE22041_DATA_OPERATIONS
Enter Employee Name: SUVETHA
Enter Basic Salary: 30000
Enter Working Days (out of 30): 28
Enter Leaves Taken: 2
Has Employee Met Targets? (true/false): TRUE

--- Employee Evaluation Report ---
Employee Name: SUVETHA
Daily Salary: ?1000.0
Salary Earned: ?28000.0
Leave Deduction: ?400.0
Net Salary: ?27600.0
Salary is Even? (mod 2): true
Full Attendance: false
Decent Working Days: true
Poor Attendance: false
Met Targets: true

>> Logical Evaluations:
Excellent Employee (Targets + No Leaves)? false
Needs HR Warning (Too Many Leaves || Low Days)? false
Award Eligible? false
? Bonus Status: Not Eligible
```

```
C:\javanew>java CCE22041_DATA_OPERATIONS
Enter Employee Name: SUVETHA
Enter Basic Salary: 30000
Enter Working Days (out of 30): 24
Enter Leaves Taken: 5
Has Employee Met Targets? (true/false): FALSE

--- Employee Evaluation Report ---
Employee Name: SUVETHA
Daily Salary: ?1000.0
Salary Earned: ?24000.0
Leave Deduction: ?1000.0
Net Salary: ?23000.0
Salary is Even? (mod 2): true
Full Attendance: false
Decent Working Days: false
Poor Attendance: false
Met Targets: false

>> Logical Evaluations:
Excellent Employee (Targets + No Leaves)? false
Needs HR Warning (Too Many Leaves || Low Days)? false
Award Eligible? false
? Bonus Status: Not Eligible
```

DATA TYPES

OBJECTIVE

To develop a Java-based **Hospital Patient Record System** that collects patient information, calculates the hospital bill based on the number of days admitted, applies insurance benefits, and demonstrates the use of **all primitive data types** in Java (byte, short, int, long, float, double, char, boolean).

OVERVIEW

The program collects patient information including:

- ID**,** contact number**,** gender**,** room charges**,** and admission details
- Uses byte and short for static info (ward number and year)
- Uses int, long for ID and contact info
- Uses float, double for financial computations
- Uses char for gender
- Uses boolean for insurance status

It then calculates the total bill with a 50% discount if the patient is insured.

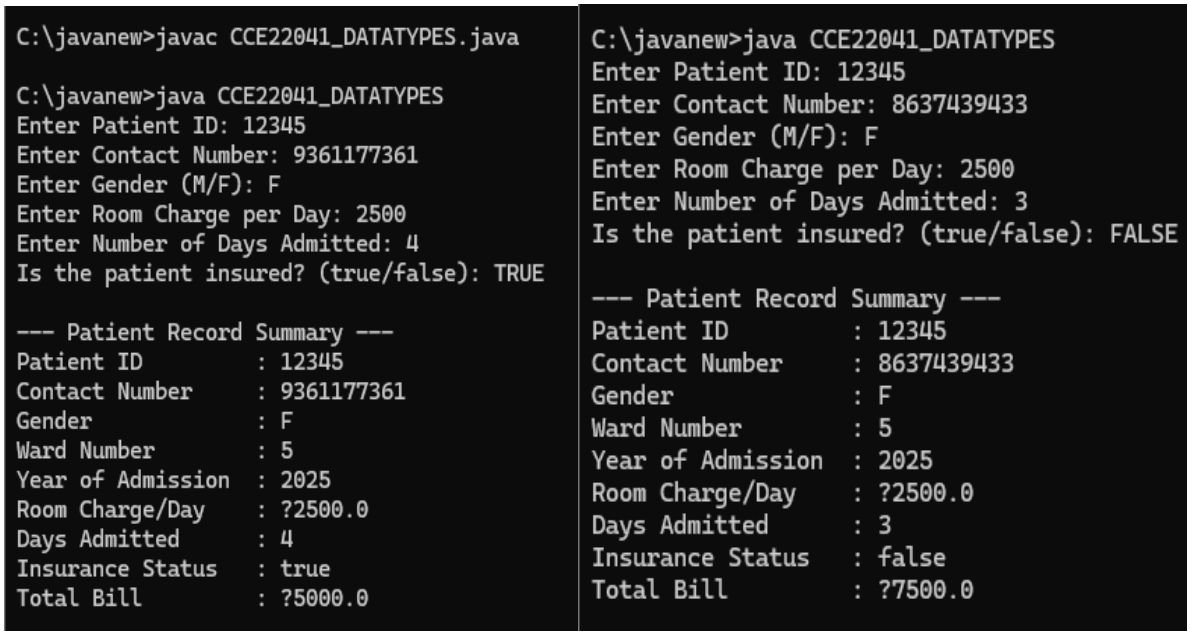
CODE:

```
import java.util.Scanner;


public class CCE22041_DATATYPES {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);



        // Integer types

        byte wardNumber = 5;            // byte

        short yearOfAdmission = 2025;    // short

        int patientID;                  // int

        long contactNumber;             // long
```

```
// Floating-point types
float roomChargePerDay;          // float
double totalBill;                // double

// Character and Boolean
char gender;                     // char
boolean isInsured;               // boolean

// Input section
System.out.print("Enter Patient ID: ");
patientID = scanner.nextInt();

System.out.print("Enter Contact Number: ");
contactNumber = scanner.nextLong();

System.out.print("Enter Gender (M/F): ");
gender = scanner.next().charAt(0);

System.out.print("Enter Room Charge per Day: ");
roomChargePerDay = scanner.nextFloat();

System.out.print("Enter Number of Days Admitted: ");
int days = scanner.nextInt();
```

```
        System.out.print("Is the patient insured? (true/false): ");
        isInsured = scanner.nextBoolean();

        // Billing logic
        totalBill = roomChargePerDay * days;
        if (isInsured) {
            totalBill *= 0.5; // 50% discount for insured patients
        }

        // Output summary
        System.out.println("\n--- Patient Record Summary ---");
        System.out.println("Patient ID         : " + patientID);
        System.out.println("Contact Number     : " + contactNumber);
        System.out.println("Gender             : " + gender);
        System.out.println("Ward Number        : " + wardNumber);
        System.out.println("Year of Admission  : " + yearOfAdmission);
        System.out.println("Room Charge/Day    : ₹" + roomChargePerDay);
        System.out.println("Days Admitted      : " + days);
        System.out.println("Insurance Status   : " + isInsured);
        System.out.println("Total Bill         : ₹" + totalBill);

        scanner.close();
    }
}
```

OUTPUTS



```
C:\javanew>javac CCE22041_DATATYPES.java

C:\javanew>java CCE22041_DATATYPES
Enter Patient ID: 12345
Enter Contact Number: 9361177361
Enter Gender (M/F): F
Enter Room Charge per Day: 2500
Enter Number of Days Admitted: 4
Is the patient insured? (true/false): TRUE

--- Patient Record Summary ---
Patient ID          : 12345
Contact Number      : 9361177361
Gender              : F
Ward Number         : 5
Year of Admission   : 2025
Room Charge/Day     : ?2500.0
Days Admitted       : 4
Insurance Status    : true
Total Bill          : ?5000.0
```

```
C:\javanew>java CCE22041_DATATYPES
Enter Patient ID: 12345
Enter Contact Number: 8637439433
Enter Gender (M/F): F
Enter Room Charge per Day: 2500
Enter Number of Days Admitted: 3
Is the patient insured? (true/false): FALSE

--- Patient Record Summary ---
Patient ID          : 12345
Contact Number      : 8637439433
Gender              : F
Ward Number         : 5
Year of Admission   : 2025
Room Charge/Day     : ?2500.0
Days Admitted       : 3
Insurance Status    : false
Total Bill          : ?7500.0
```

CONTROL STATEMENTS

OBJECTIVE

To demonstrate the use of all control statements (if, if-else, nested if, if-else-if ladder, switch-case, for, while, do-while, break, continue) through a real-world ATM Simulation application.

OVERVIEW

ATM Simulation application

- Verified PIN using if, if-else, and nested if.
- Displayed a menu using do-while.
- Handled user choices using switch-case with if-else-if ladder.
- Processed deposits and withdrawals.
- Displayed last transactions using for loop with break and continue.
- Demonstrated while loop for retry logic (as demo).
- Exited the loop on choice = 5.

CODE

```
import java.util.Scanner;

public class CCE22041_CONTROL_STATEMENTS {

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

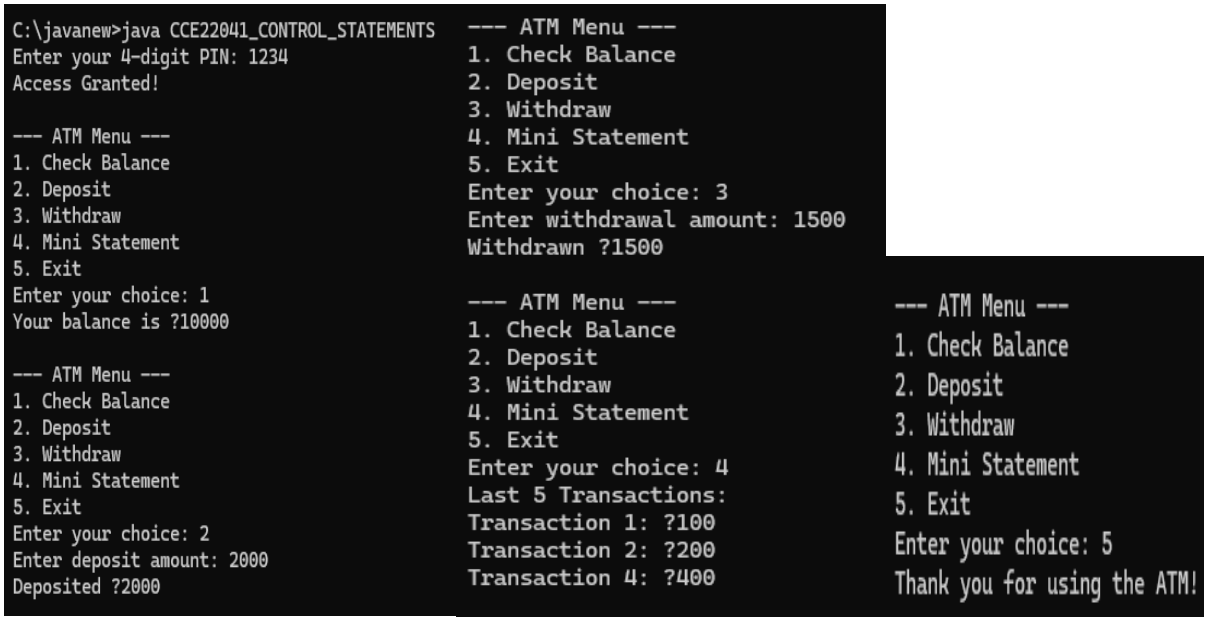
```
Scanner scanner = new Scanner(System.in);
int correctPin = 1234;
int balance = 10000;

// --- if, if-else, nested if ---
System.out.print("Enter your 4-digit PIN: ");
int enteredPin = scanner.nextInt();

if (enteredPin == correctPin) {
    System.out.println("Access Granted!");
} else {
    System.out.println("Wrong PIN. Access Denied.");
    return; // Exit the program
}

// --- do-while loop: ATM menu repeats until user chooses to exit ---
int choice;
do {
    System.out.println("\n--- ATM Menu ---");
    System.out.println("1. Check Balance");
    System.out.println("2. Deposit");
    System.out.println("3. Withdraw");
    System.out.println("4. Mini Statement");
    System.out.println("5. Exit");
```

```
System.out.print("Enter your choice: ");
choice = scanner.nextInt();

// --- switch-case + if-else ladder inside ---
switch (choice) {
    case 1:
        System.out.println("Your balance is ₹" + balance);
        break;

    case 2:
        System.out.print("Enter deposit amount: ");
        int deposit = scanner.nextInt();
        if (deposit <= 0) {
            System.out.println("Invalid amount.");
            break;
        }
        balance += deposit;
        System.out.println("Deposited ₹" + deposit);
        break;

    case 3:
        System.out.print("Enter withdrawal amount: ");
        int withdraw = scanner.nextInt();
        if (withdraw <= 0) {
```

```
            System.out.println("Invalid amount.");
        } else if (withdraw > balance) {
            System.out.println("Insufficient funds.");
        } else {
            balance -= withdraw;
            System.out.println("Withdrawn ₹" + withdraw);
        }
        break;

    case 4:
        // --- for loop with break and continue ---
        System.out.println("Last 5 Transactions:");
        for (int i = 1; i <= 5; i++) {
            if (i == 3) continue; // skip 3rd record
            if (i == 5) break;    // stop early
            System.out.println("Transaction " + i + ": ₹" + (100 * i));
        }
        break;

    case 5:
        System.out.println("Thank you for using the ATM!");
        break;

    default:
```

```
                System.out.println("Invalid choice.");
            }

            // --- while loop example (just for demo) ---
            int retry = 0;
            while (retry < 0) { // won't run but added to demo
                System.out.println("Retry attempt: " + retry);
                retry++;
            }

        } while (choice != 5); // end when user chooses to exit

        scanner.close();
    }
}
```

OUTPUTS



ACCESS MODIFIERS

OBJECTIVE

This program demonstrates the use of **all Access Modifiers** (public, private, protected, default) and **Non-Access Modifiers** (static, final, abstract) in a **University System** application.

OVERVIEW:

University System

- Models a university with departments
- Uses an abstract class for university information
- Uses a subclass for department-specific details
- Shows object creation, static counters, and inheritance

CODE :

```
// Abstract class with protected and final members

abstract class University {

    protected String universityName = "National University";  // protected

    final int establishedYear = 1950;                    // final



    public void showUniversityInfo() {

        System.out.println("University: " + universityName);

        System.out.println("Established in: " + establishedYear);

    }



    // Abstract method to be implemented by subclass

    abstract void showDepartmentInfo();

}

class Department extends University {

    public String departmentName;                // public
```

```
    private int departmentCode;                    // private
    protected static int totalDepartments = 0;      // static & protected
  // Constructor
    public Department(String deptName, int deptCode) {
        this.departmentName = deptName;
        this.departmentCode = deptCode;
        totalDepartments++;  // static variable updated
    }
 // Overriding abstract method
    @Override
    void showDepartmentInfo() {
        System.out.println("Department: " + departmentName);
        System.out.println("Department Code: " + departmentCode);
    }
   // Static method to show total departments
    public static void showTotalDepartments() {
        System.out.println("Total Departments: " + totalDepartments);
    }
}
// Main class with default (package-private) access
class UniversitySystem {
    public static void main(String[] args) {
        Department d1 = new Department("Computer Science", 101);
        Department d2 = new Department("Mechanical Engineering", 102);
```

```
        d1.showUniversityInfo();
        d1.showDepartmentInfo();
         System.out.println();
        d2.showUniversityInfo();
        d2.showDepartmentInfo();
    // Static method call
        Department.showTotalDepartments();
     }
}
```

OUTPUT:

```
C:\javanew>javac CCE22041_ACCESS_MODIFIERS.java

C:\javanew>java UniversitySystem
University: National University
Established in: 1950
Department: Computer Science
Department Code: 101

University: National University
Established in: 1950
Department: Mechanical Engineering
Department Code: 102
Total Departments: 2
```