TASK 6: Object-Oriented Modeling – Bank Account System .

1. Introduction

This task is based on **Object-Oriented Programming (OOP)** concepts in Java.
We design a simple **Bank Account System** where each bank account is represented as an object.
The main goal is to understand **encapsulation, data hiding, constructors, classes, objects, and object interaction**.

2. Problem Understanding

We need to create a banking system that can:

- Store account details securely
- Allow deposit and withdrawal of money
- Validate transactions using business rules
- Maintain transaction history
- Create and manage multiple bank accounts

3. Class Design: BankAccount

Why we use a class?

A **class** is a blueprint for creating objects.
Here, BankAccount is a blueprint that defines how a bank account should look and behave.

4. Data Members (Variables)

private String accountHolderName;

private int accountNumber;

private double balance;

5. Constructor:- public BankAccount(String name, int accNo, double initialBalance) {

accountHolderName = name;

accountNumber = accNo;

balance = initialBalance;

}

Role of Constructor:

- Automatically called when an object is created
- Initializes account details
- Ensures every account starts with valid data

6. Encapsulation Using Getters:-

public double getBalance() {

return balance;

}

7. Deposit Method:- public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

8. Withdrawal Method:- public void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

}

}

9. Transaction History:-

private ArrayList<String> transactionHistory;

10. Object Creation (Main Class):-

BankAccount acc1 = new BankAccount("Akash Kumar", 101, 5000);

BankAccount acc2 = new BankAccount("Rahul Sharma", 102, 3000);

11. Object Interaction:-

acc1.deposit(2000);

acc1.withdraw(1500);

12. Key OOP Concepts Used

Encapsulation

Wrapping data and methods together and restricting direct access using private.

Data Hiding

Sensitive data is hidden from outside access.

Class

Blueprint that defines structure and behavior.

Object

Real instance of a class with actual data.

Constructor

Initializes objects at the time of creation.

13. Object vs Class:-

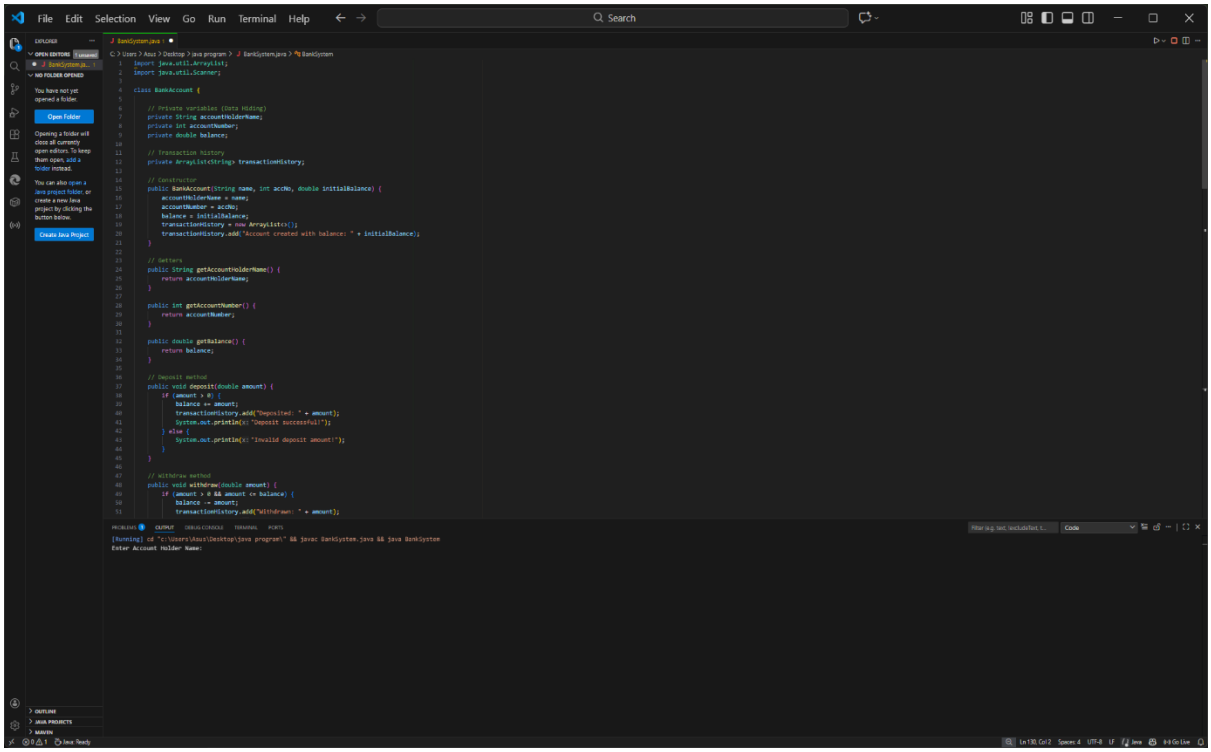
Class	**Object**
Blueprint	**Instance**

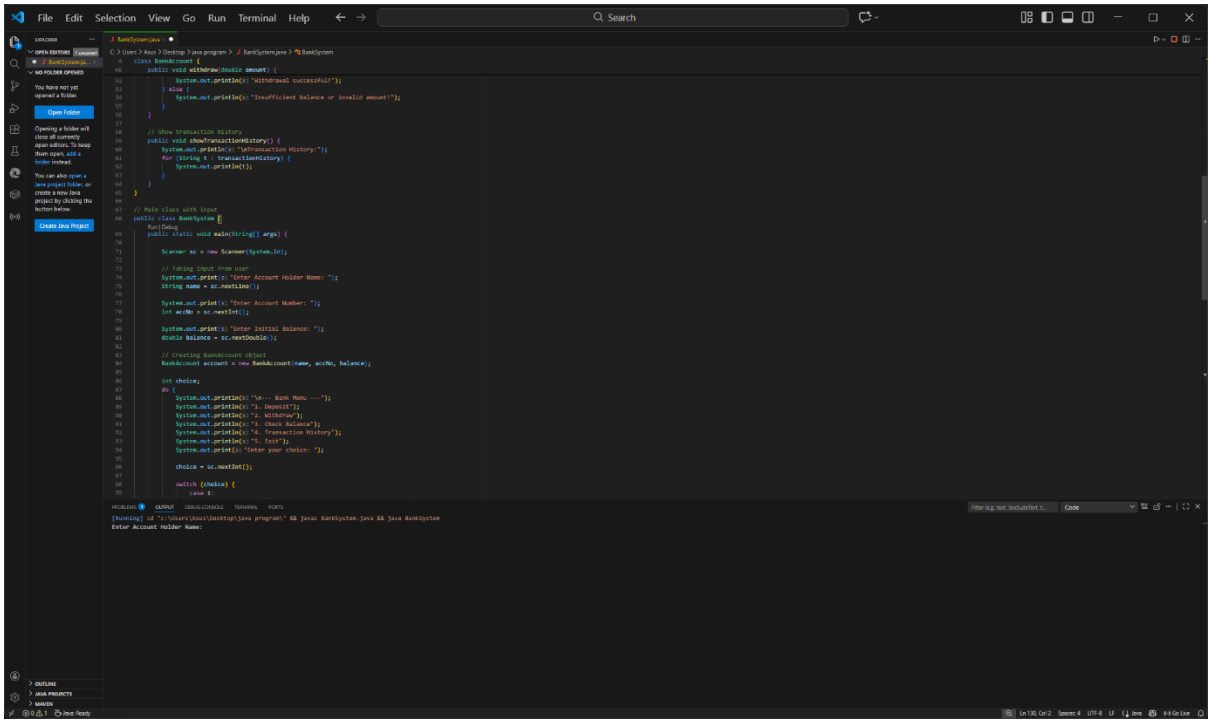
Class	Object
Logical	Physical
Defines structure	Holds actual data

14. Advantages of This Design

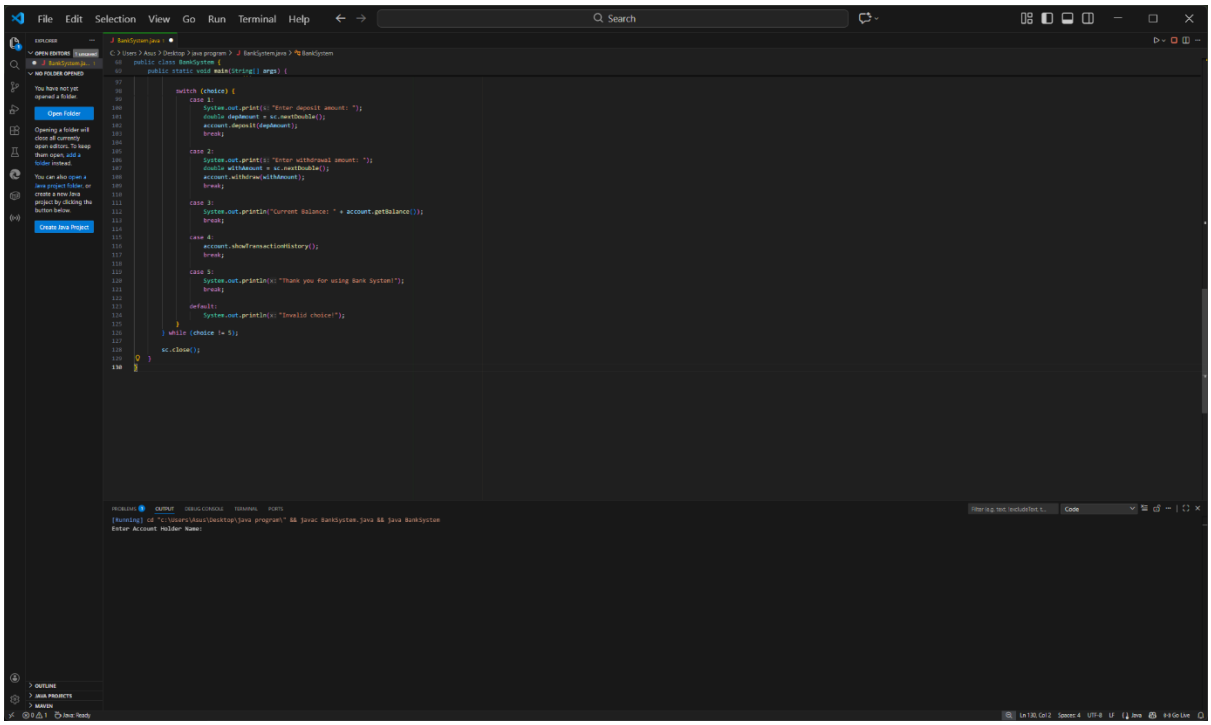
- Secure data handling
- Easy maintenance
- Real-world banking simulation
- Reusable and scalable code
- Interview-ready OOP implementation

Code:-





```
File Edit Selection View Go Run Terminal Help
Search
BankSystem.java
C: > Users > Asus > Desktop > java program > BankSystem.java > BankSystem
class BankAccount {
    public void withdraw(double amount) {
        if (amount > 0 && balance >= amount) {
            balance -= amount;
            transactionHistory.add("Withdrawal: " + amount);
            System.out.println("Withdrawal successful!");
        } else {
            System.out.println("Insufficient balance or invalid amount!");
        }
    }

    // Show transaction history
    public void showTransactionHistory() {
        System.out.println("\nTransaction History:");
        for (String t : transactionHistory) {
            System.out.println(t);
        }
    }
}

// Main class with input
public class BankSystem {
    Run | Debug
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Taking input from user
        System.out.print("Enter Account Holder Name: ");
        String name = sc.nextLine();

        System.out.print("Enter Account Number: ");
        int accNo = sc.nextInt();

        System.out.print("Enter Initial Balance: ");
        double balance = sc.nextDouble();

        // Creating BankAccount object
        BankAccount account = new BankAccount(name, accNo, balance);

        int choice;
        do {
            System.out.println("\n--- Bank Menu ---");
            System.out.println("1. Deposit");
            System.out.println("2. Withdraw");
            System.out.println("3. Check Balance");
            System.out.println("4. Transaction History");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            choice = sc.nextInt();

            switch (choice) {
                case 1:
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] cd "c:\Users\Asus\Desktop\java program\" && javac BankSystem.java && java BankSystem
Enter Account Holder Name:
OUTLINE
JAVA PROJECTS
MAVEN
```



```
File Edit Selection View Go Run Terminal Help
Search
BankSystem.java
C: > Users > Asus > Desktop > java program > BankSystem.java > BankSystem
public class BankSystem {
    public static void main(String[] args) {

            switch (choice) {
                case 1:
                    System.out.print("Enter deposit amount: ");
                    double depAmount = sc.nextDouble();
                    account.deposit(depAmount);
                    break;

                case 2:
                    System.out.print("Enter withdrawal amount: ");
                    double withAmount = sc.nextDouble();
                    account.withdraw(withAmount);
                    break;

                case 3:
                    System.out.println("Current Balance: " + account.getBalance());
                    break;

                case 4:
                    account.showTransactionHistory();
                    break;

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

                default:
                    System.out.println("Invalid choice!");
            }
        } while (choice != 5);

        sc.close();
    }
}
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[Running] cd "c:\Users\Asus\Desktop\java program\" && javac BankSystem.java && java BankSystem
Enter Account Holder Name:
OUTLINE
JAVA PROJECTS
MAVEN
```