

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:-

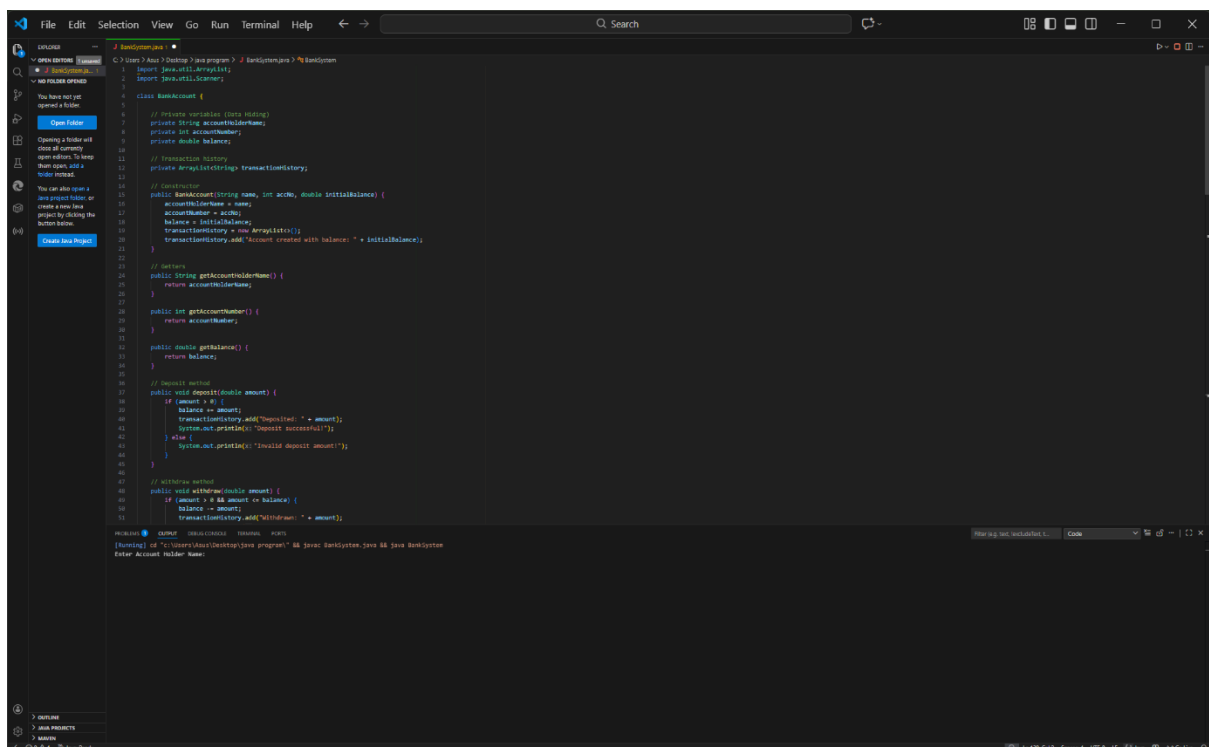
Class	Object
Blueprint	Instance

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:-



```

1  import java.util.ArrayList;
2  import java.util.Scanner;
3
4  class BankAccount {
5
6      // private attributes (Data Hiding)
7      private String accountHolderName;
8      private int accountNumber;
9      private double balance;
10
11      // Transaction History
12      private ArrayList<String> transactionHistory;
13
14      // constructor
15      public BankAccount(String name, int acNo, double initialBalance) {
16          accountHolderName = name;
17          accountNumber = acNo;
18          balance = initialBalance;
19          transactionHistory = new ArrayList<>();
20          transactionHistory.add("Account created with balance: " + initialBalance);
21      }
22
23      // getters
24      public String getAccountHolderName() {
25          return accountHolderName;
26      }
27
28      public int getAccountNumber() {
29          return accountNumber;
30      }
31
32      public double getBalance() {
33          return balance;
34      }
35
36      // Deposit method
37      public void deposit(double amount) {
38          if (amount > 0) {
39              balance += amount;
40              transactionHistory.add("Deposit: " + amount);
41              System.out.println("Deposit successful");
42          } else {
43              System.out.println("Invalid deposit amount");
44          }
45      }
46
47      // Withdrawal method
48      public void withdraw(double amount) {
49          if (amount > 0 && amount <= balance) {
50              balance -= amount;
51              transactionHistory.add("Withdrawal: " + amount);
52          }
53      }
54
55      // Main method
56      public static void main(String[] args) {
57          Scanner scanner = new Scanner(System.in);
58          System.out.println("Enter Account Holder Name:");
59          String name = scanner.nextLine();
60          System.out.println("Enter Account Number:");
61          int acNo = scanner.nextInt();
62          System.out.println("Enter Initial Balance:");
63          double initialBalance = scanner.nextDouble();
64          BankAccount account = new BankAccount(name, acNo, initialBalance);
65          while (true) {
66              System.out.println("\n1. Deposit\n2. Withdraw\n3. Get Balance\n4. Get Account Number\n5. Get Account Holder Name\n6. Exit");
67              int choice = scanner.nextInt();
68              switch (choice) {
69                  case 1:
69                      System.out.println("Enter Amount to Deposit:");
69                      double depositAmount = scanner.nextDouble();
69                      account.deposit(depositAmount);
69                      break;
69                  case 2:
69                      System.out.println("Enter Amount to Withdraw:");
69                      double withdrawAmount = scanner.nextDouble();
69                      account.withdraw(withdrawAmount);
69                      break;
69                  case 3:
69                      System.out.println("Current Balance: " + account.getBalance());
69                      break;
69                  case 4:
69                      System.out.println("Account Number: " + account.getAccountNumber());
69                      break;
69                  case 5:
69                      System.out.println("Account Holder Name: " + account.getAccountHolderName());
69                      break;
69                  case 6:
69                      System.out.println("Exiting...");
69                      break;
69                  default:
69                      System.out.println("Invalid choice");
69                      break;
69              }
69          }
69      }
69  }

```

```
1 import java.util.Scanner;
2
3 class BankAccount {
4     public void withdraw(double amount) {
5         System.out.println("Withdrawal successful");
6         // size {
7         System.out.println("Insufficient balance or invalid amount");
8     }
9 }
10
11 // Show transaction history
12 public void showTransactionHistory() {
13     System.out.println("Transaction history:");
14     for (String t : transactionHistory) {
15         System.out.println(t);
16     }
17 }
18
19 // Main class with input
20 public class BankSystem {
21     // Main method
22     public static void main(String[] args) {
23         Scanner sc = new Scanner(System.in);
24
25         // Taking input from user
26         System.out.print("Enter Account Holder Name: ");
27         String name = sc.nextLine();
28
29         System.out.print("Enter Account Number: ");
30         int acNo = sc.nextInt();
31
32         System.out.print("Enter Initial Balance: ");
33         double balance = sc.nextDouble();
34
35         // Creating BankAccount object
36         BankAccount account = new BankAccount(name, acNo, balance);
37
38         int choice;
39         do {
40             System.out.println("\n--- Bank Menu ---");
41             System.out.println("1. Deposit");
42             System.out.println("2. Withdraw");
43             System.out.println("3. Check Balance");
44             System.out.println("4. Transaction History");
45             System.out.println("5. Exit");
46             System.out.print("Enter your choice: ");
47
48             choice = sc.nextInt();
49
50             switch (choice) {
51                 case 1:
```

```
52                 case 2:
53                     System.out.print("Enter deposit amount: ");
54                     double deposit = sc.nextDouble();
55                     account.deposit(deposit);
56                     break;
57                 case 3:
58                     System.out.print("Enter withdrawal amount: ");
59                     double withdrawal = sc.nextDouble();
60                     account.withdraw(withdrawal);
61                     break;
62                 case 4:
63                     System.out.print("Current Balance: " + account.getBalance());
64                     break;
65                 case 5:
66                     account.showTransactionHistory();
67                     break;
68                 case 6:
69                     System.out.print("Thank you for using Bank System");
70                     break;
71                 default:
72                     System.out.print("Invalid choice");
73             }
74         } while (choice != 5);
75         sc.close();
76     }
77 }
```