UITS

Future will be better than thy past

An initiative of PHP Family

University of Information Technology & Sciences

OOP Lab – Project Report

Course Title : Object-Oriented Programming Language Lab

Course Code : CSE0613122

Topic : Bank Management System

Submitted to:

Teacher's Name : Md. Ismail

Designation : Lecturer

Submitted by:

Student Name:	Student Roll / ID:
Khandakar Borhan Uddin	4323100051010 08
Md. Ashraful Haque Zani	4324100051010 75
Gaus Saraf Murady	04324100051010 88

Semester : Autumn, 2024

Batch : 55

Department : CSE

Date of Submission: 09.12.24

Project Report for Bank Account Management System

Project Overview:

The Bank Account Management System is a Java-based program that simulates banking operations for two types of accounts: Savings Account and Current Account. It provides functionalities such as depositing money, displaying balance, computing and depositing interest (for savings accounts), withdrawing money, and ensuring minimum balance compliance (for current accounts).

The program utilizes inheritance to model the relationship between generic accounts (Account) and their specific types (SavAcct and CurrAcct).

Objectives:

Model real-world banking features using object-oriented programming (OOP) concepts. Use inheritance to create specialized classes for savings and current accounts. Provide interactive functionality to handle customer transactions. Implement error handling to ensure data validity and user-friendly error reporting.

Features:

1. Savings Account	Compound interest computation and deposit. Deposit and withdrawal capabilities. No minimum balance requirement.
2. Current Account	Minimum balance enforcement with penalty for non-compliance. Deposit and withdrawal capabilities. No interest computation.
3. Generic Functionalities	Balance display. Transaction logging through console messages. Exception handling for invalid inputs or operations.

Class Structure and Methods:

1. Class: Account	Purpose:	• Acts as a base class for all types of accounts. Stores common attributes and methods.
	Attributes:	• customerName: Name of the account holder. • accountNumber: Unique identifier for the account. • accountType: Type of the account (Savings/Current). • balance: Current account balance.
	Methods:	• initialize(): Initializes account details. • deposit(double amount): Adds money to the balance. Throws exception for invalid inputs. • displayBalance(): Displays the current balance. • withdraw(double amount): Deducts money from the balance. Handles insufficient funds.
2. Class: SavAcct	Purpose:	• Represents a savings account.
	Attributes:	• interestRate: Fixed annual interest rate (4%).
	Methods:	• computeAndDepositInterest(): Computes interest on the balance and adds it.
3. Class: CurrAcct	Purpose:	• Represents a current account.
	Attributes:	• minimumBalance: Required minimum balance (500). • penalty: Penalty for falling below minimum balance (50).
	Methods:	• checkMinimumBalance(): Checks if balance meets the minimum requirement and applies penalty if not.
4. Class: BankAccountManagement	Purpose:	• Entry point for the program and contains the main logic for user interaction.
	Methods:	• manageSavingsAccount(): Handles savings account operations via a menu-driven interface. • manageCurrentAccount(): Handles current account operations via a menu-driven interface.

Implementation Details:

Key Concepts Used	1. Inheritance:	SavAcct and CurrAcct extend Account to inherit common attributes and methods.
	2. Encapsulation:	Class attributes are protected via public methods.
	3. Exception Handling:	Ensures user inputs and operations are valid. Provides meaningful error messages for invalid cases.
Menu-Driven Interface	The program uses a menu system for each account type, allowing users to:	1. Deposit money. 2. Display the current balance. 3. Compute and deposit interest (savings account only). 4. Check minimum balance and apply penalties (current account only). 5. Withdraw money.
Input Validation	Deposit Amount:	Must be positive.
	Withdrawal Amount:	Must be positive and less than or equal to the current balance.

Code:

```
import java.util.Scanner;

// Base class representing a generic bank account
class Account {
    String customerName;   // Name of the account holder
    long accountNumber;    // Unique identifier for the account
    String accountType;    // Type of account (Savings/Current)
    double balance;        // Current balance in the account

    // Method to initialize account details
    void initialize(String name, long accNo, String type, double initialBalance) {
        this.customerName = name;
        this.accountNumber = accNo;
        this.accountType = type;
        this.balance = initialBalance;
    }

    // Method to deposit money into the account
    void deposit(double amount) throws IllegalArgumentException {
        if (amount <= 0) {
            throw new IllegalArgumentException("Deposit amount must be positive.");
        }
        balance += amount; // Update the balance
        System.out.println("Deposit successful! New balance: " + balance);
    }

    // Method to display the current balance
    void displayBalance() {
        System.out.println("Account Balance: " + balance);
    }

    // Method to withdraw money from the account
    void withdraw(double amount) throws IllegalArgumentException, IllegalStateException {
        if (amount <= 0) {
            throw new IllegalArgumentException("Withdrawal amount must be positive.");
        }
        if (amount > balance) {
            throw new IllegalStateException("Insufficient balance for withdrawal.");
        }
        balance -= amount; // Deduct the amount from balance
        System.out.println("Withdrawal successful! New balance: " + balance);
    }
}

// Subclass representing a savings account
class SavAcct extends Account {
    final double interestRate = 0.04; // Annual interest rate (4%)

    // Method to compute and deposit interest
    void computeAndDepositInterest() {
        double interest = balance * interestRate; // Calculate interest
        balance += interest; // Add interest to balance
        System.out.println("Interest computed and deposited! New balance: " + balance);
    }
}

// Subclass representing a current account
class CurrAcct extends Account {
    final double minimumBalance = 500; // Minimum balance requirement
    final double penalty = 50;         // Penalty for falling below minimum balance

    // Method to check and enforce the minimum balance requirement
    void checkMinimumBalance() {
        if (balance < minimumBalance) {
            balance -= penalty; // Deduct penalty if balance is below minimum
            System.out.println("Balance below minimum! Penalty of " + penalty + " imposed. New balance: " + balance);
        } else {
            System.out.println("Minimum balance maintained.");
        }
    }
}

// Main class to manage the bank account system
public class BankAccountManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in); // Input scanner
        SavAcct savingsAccount = new SavAcct();
        CurrAcct currentAccount = new CurrAcct();

        try {
            // Prompt user to enter account details
            System.out.println("Enter account type (Savings/Current): ");
            String accountType = scanner.next();

            System.out.println("Enter customer name: ");
            String name = scanner.next();

            System.out.println("Enter account number: ");
            long accountNumber = scanner.nextLong();

            System.out.println("Enter initial balance: ");
            double initialBalance = scanner.nextDouble();

            // Handle account type and initialize appropriate object
            switch (accountType.toLowerCase()) {
                case "savings": {
                    savingsAccount.initialize(name, accountNumber, accountType, initialBalance);
                    manageSavingsAccount(scanner, savingsAccount); // Manage savings account operations
                    break;
                }
```

```
            case "current": {
                currentAccount.initialize(name,
accountNumber, accountType, initialBalance);
                manageCurrentAccount(scanner,
currentAccount); // Manage current account operations
                break;
            }
            default: {
                System.out.println("Invalid account type!");
// Handle invalid input
                break;
            }
          }
       } catch (Exception e) {
          // Handle any unexpected errors
          System.out.println("Error: " + e.getMessage());
       } finally {
          scanner.close(); // Close the scanner to free
resources
       }
    }

    // Method to manage savings account operations
    static void manageSavingsAccount(Scanner scanner,
SavAcct account) {
       boolean exit = false; // Flag to exit the menu
       while (!exit) {
          try {
             // Display menu options
             System.out.println("\n1. Deposit\n2. Display
Balance\n3. Compute Interest\n4. Withdraw\n5. Exit");
             System.out.println("Enter your choice: ");
             int choice = scanner.nextInt();

             // Handle menu options
             switch (choice) {
                case 1: {
                   System.out.println("Enter amount to
deposit: ");
                   double amount = scanner.nextDouble();
                   account.deposit(amount);
                   break;
                }
                case 2: {
                   account.displayBalance();
                   break;
                }
                case 3: {
                   account.computeAndDepositInterest();
                   break;
                }
                case 4: {
                   System.out.println("Enter amount to
withdraw: ");
                   double amount = scanner.nextDouble();
                   account.withdraw(amount);
                   break;
                }
                case 5: {
                   exit = true; // Exit the menu
                   break;
                }
                default: {
                   System.out.println("Invalid choice!");
                }
             }
          } catch (Exception e) {
             // Handle any menu-specific errors
             System.out.println("Error: " + e.getMessage());
          }
       }
    }

    // Method to manage current account operations
    static void manageCurrentAccount(Scanner scanner,
CurrAcct account) {
       boolean exit = false; // Flag to exit the menu
       while (!exit) {
          try {
             // Display menu options
             System.out.println("\n1. Deposit\n2. Display
Balance\n3. Check Minimum Balance\n4. Withdraw\n5.
Exit");
             System.out.println("Enter your choice: ");
             int choice = scanner.nextInt();

             // Handle menu options
             switch (choice) {
                case 1: {
                   System.out.println("Enter amount to
deposit: ");
                   double amount = scanner.nextDouble();
                   account.deposit(amount);
                   break;
                }
                case 2: {
                   account.displayBalance();
                   break;
                }
                case 3: {
                   account.checkMinimumBalance();
                   break;
                }
                case 4: {
                   System.out.println("Enter amount to
withdraw: ");
                   double amount = scanner.nextDouble();
                   account.withdraw(amount);
                   break;
                }
                case 5: {
                   exit = true; // Exit the menu
                   break;
                }
                default: {
                   System.out.println("Invalid choice!");
                }
             }
          } catch (Exception e) {
             // Handle any menu-specific errors
             System.out.println("Error: " + e.getMessage());
          }
       }
    }
}
```

Sample Interaction

Note: Interest Rate is 4% and Minimum Balance is 500 (Penalty for lower Balance is 50)

Enter account type (Savings/Current): Savings Enter customer name: Gaus Enter account number: 12 Enter initial balance: 1000	
(default interface)	(interface with user input)
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 1 Enter amount to deposit: 50 Deposit successful! New balance: 1050.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 2 Account Balance: 1050.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 3 Interest computed and deposited! New balance: 1092.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 4 Enter amount to withdraw: 50 Withdrawal successful! New balance: 1042.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 5 (Program Finishes)

Enter account type (Savings/Current): Current Enter customer name: Borhan Enter account number: 13 Enter initial balance: 500	
(default interface)	(interface with user input)
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 1 Enter amount to deposit: 50 Deposit successful! New balance: 550.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 2 Account Balance: 550.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 4 Enter amount to withdraw: 100 Withdrawal successful! New balance: 450.0 (to show penalty)
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 3 Balance below minimum! Penalty of 50.0 imposed. New balance: 400.0
1. Deposit 2. Display Balance 3. Compute Interest 4. Withdraw 5. Exit	Enter your choice: 5 (Program Finishes)

Exception Handling

Invalid Deposit or Withdrawal Amounts:	Throws IllegalArgumentException for non-positive amounts.
Insufficient Balance for Withdrawal:	Throws IllegalStateException.
General Input Errors:	Catches Exception for unexpected issues during input parsing.

Limitations

No persistence:	Account data resets after program termination.
No multi-user support:	Only one account can be managed per session.
Limited features:	Real-world banking operations like account creation and deletion are not implemented.

Future Enhancements

1. Database Integration:	Store account data in a database for persistence and multi-user support.
2. GUI Implementation:	Replace the console-based interface with a graphical user interface.
3. Additional Account Types:	Add support for more account types (e.g., Fixed Deposit, Business Accounts).
4. Advanced Features:	Include functionality for cheque issuance and processing.