

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 | 432310005101008 |
| Md. Ashraful Haque Zani | 432410005101075 |
| Gaus Saraf Murady | 0432410005101088 |

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.\*;  interface Authenticatable { boolean authenticate(String password); }  abstract class Account { String name, accType; int accNum, accBalance; private static final Set<Integer> usedAccountNumbers = new HashSet<>();  // Constructor to initialize account details void initialize(String name, String accType, int accBalance) {* [*this.name*](http://this.name/) *= name; this.accType = accType; this.accBalance = accBalance; this.accNum = generateAccountNumber(); }  // Method to generate unique account numbers protected static int generateAccountNumber() { int accNum; do { accNum = (int) ((Math.random() \* 9000) + 1000); // Generate a 4-digit number } while (usedAccountNumbers.contains(accNum)); // Ensure it's unique usedAccountNumbers.add(accNum); return accNum; }  // Abstract methods that need to be implemented by subclasses abstract void displayDetails(); abstract void deposit(int amount); abstract void withdraw(int amount); }  class SavingsAccount extends Account implements Authenticatable { private String password; private static final double INTEREST\_RATE = 0.03; // 3% annual interest  void initializeSavings(String name, String password, int initialBalance) { super.initialize(name, "Savings", initialBalance); this.password = password; }  @Override public boolean authenticate(String inputPassword) { return this.password.equals(inputPassword); }  void computeAndDepositInterest() { int interest = (int) (accBalance \* INTEREST\_RATE); accBalance += interest; System.out.println("Interest of " + interest + " added. New balance: " + accBalance); }  @Override void displayDetails() { System.out.println("Account Type: Savings"); System.out.println("Name: " + name); System.out.println("Account Number: " + accNum); System.out.println("Balance: " + accBalance); }  @Override void deposit(int amount) { accBalance += amount; System.out.println("Deposit successful. New balance: " + accBalance); }  @Override void withdraw(int amount) { if (amount <= accBalance) { accBalance -= amount; System.out.println("Withdrawal successful. New balance: " + accBalance); } else { System.out.println("Insufficient funds."); } } }  class CurrentAccount extends Account implements Authenticatable { private String password; private static final int MIN\_BALANCE = 1000; private static final int PENALTY = 100;  void initializeCurrent(String name, String password, int initialBalance) { super.initialize(name, "Current", initialBalance); this.password = password; }  @Override public boolean authenticate(String inputPassword) { return this.password.equals(inputPassword); }  void checkMinimumBalance() { if (accBalance < MIN\_BALANCE) { accBalance -= PENALTY; System.out.println("Minimum balance not maintained. Penalty of " + PENALTY + " imposed. New balance: " + accBalance); } }  @Override void displayDetails() { System.out.println("Account Type: Current"); System.out.println("Name: " + name); System.out.println("Account Number: " + accNum); System.out.println("Balance: " + accBalance); }  @Override void deposit(int amount) { accBalance += amount; System.out.println("Deposit successful. New balance: " + accBalance); }  @Override void withdraw(int amount) { if (amount <= accBalance) { accBalance -= amount; System.out.println("Withdrawal successful. New balance: " + accBalance); } else { System.out.println("Insufficient funds."); } } }  public class BankOOPproject { public static void main(String[] args) { List<Account> accounts = new ArrayList<>(); Scanner in = new Scanner(System.in); boolean quit = false;  while (!quit) { try { System.out.println("1. Create Savings Account"); System.out.println("2. Create Current Account"); System.out.println("3. Deposit Money"); System.out.println("4. Withdraw Money"); System.out.println("5. Check Balance"); System.out.println("6. Display Account Details"); System.out.println("7. Compute Interest (Savings Only)"); System.out.println("8. Check Minimum Balance (Current Only)"); System.out.println("0. Quit"); System.out.print("Enter Your Choice: "); int choice = in.nextInt();  switch (choice) { case 1: { SavingsAccount savings = new SavingsAccount(); System.out.print("Enter Name: "); String name = in.next(); System.out.print("Set Password: "); String password = in.next(); System.out.print("Enter Initial Balance: "); int balance = in.nextInt(); savings.initializeSavings(name, password, balance); accounts.add(savings); System.out.println("Savings Account created successfully."); System.out.println("Account Number: " + savings.accNum); break; } case 2: { CurrentAccount current = new CurrentAccount(); System.out.print("Enter Name: "); String name = in.next(); System.out.print("Set Password: "); String password = in.next(); System.out.print("Enter Initial Balance: "); int balance = in.nextInt(); current.initializeCurrent(name, password, balance); accounts.add(current); System.out.println("Current Account created successfully."); System.out.println("Account Number: " + current.accNum); break; } case 3: { Account account = findAccount(accounts, in); if (account != null) { System.out.print("Enter Amount to Deposit: "); int amount = in.nextInt(); account.deposit(amount); } break; } case 4: { Account account = authenticateAccount(accounts, in); if (account != null) { System.out.print("Enter Amount to Withdraw: "); int amount = in.nextInt(); account.withdraw(amount); } break; } case 5: { Account account = authenticateAccount(accounts, in); if (account != null) { System.out.println("Balance: " + account.accBalance); } break; } case 6: { Account account = authenticateAccount(accounts, in); if (account != null) { account.displayDetails(); } break; } case 7: { Account account = authenticateAccount(accounts, in); if (account instanceof SavingsAccount) { SavingsAccount savingsAccount = (SavingsAccount) account; savingsAccount.computeAndDepositInterest(); } else { System.out.println("This feature is for Savings Accounts only."); } break; } case 8: { Account account = authenticateAccount(accounts, in); if (account instanceof CurrentAccount) { CurrentAccount currentAccount = (CurrentAccount) account; currentAccount.checkMinimumBalance(); } else { System.out.println("This feature is for Current Accounts only."); } break; } case 0: quit = true; break; default: System.out.println("Invalid choice. Please try again."); break; } } catch (InputMismatchException e) { System.out.println("Invalid input. Please enter a number."); in.next(); // Clear the invalid input } } }  private static Account findAccount(List<Account> accounts, Scanner in) { System.out.print("Enter Account Number: "); int accNum = in.nextInt(); for (Account account : accounts) { if (account.accNum == accNum) { return account; } } System.out.println("Account not found."); return null; }  private static Account authenticateAccount(List<Account> accounts, Scanner in) { Account account = findAccount(accounts, in); if (account == null) return null;  if (account instanceof Authenticatable) { System.out.print("Enter Password: "); String password = in.next(); if (!((Authenticatable) account).authenticate(password)) { System.out.println("Authentication failed. Incorrect password."); return null; } } return account; } }* |

**Sample Interaction**

|  |  |
| --- | --- |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter your choice: 1 (Savings Account Menu)  Enter name: John  Set Password: 12345  Enter initial balance: 1000 |
| Savings Account created successfully.  Account Number: 4013 (RNG) |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter your choice: 2 (Current Account Menu)  Enter name: Jane  Set Password: 54321  Enter initial balance: 450 |
| Current Account created successfully.  Account Number: 9665 (RNG) |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter your choice: 3  Enter Account Number: 4013  Enter Amount to Deposit: 500  Deposit successful. New balance: 1500 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 4  Enter Account Number: 4013  Enter Password: 12345  Enter Amount to Withdraw: 1400  Withdrawal successful. New balance: 100 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 4  Enter Account Number: 9665  Enter Password: 54321  Enter Amount to Withdraw: 50  Withdrawal successful. New balance: 400 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 5  Enter Account Number: 4013  Enter Password: 12345  Balance: 100 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 6  Enter Account Number: 4013  Enter Password: 12345  Account Type: Savings  Name: John  Account Number: 4013  Balance: 100 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 7  Enter Account Number: 4013  Enter Password: 12345  Interest of 3 added. New balance: 103 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 8 (Minimum Balance is 1000)  Enter Account Number: 9665  Enter Password: 54321  Minimum balance not maintained. Penalty of 100 imposed. New balance: 300 |
| 1. Create Savings Account  2. Create Current Account  3. Deposit Money  4. Withdraw Money  5. Check Balance  6. Display Account Details  7. Compute Interest (Savings Only)  8. Check Minimum Balance (Current Only)  0. Quit | Enter Your Choice: 0  (Program Finished) |

**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. (Savings and Current account can be managed separately but there cannot be multiple accounts for the same type) |
| 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. |