Southeast University

Department of Computer Science and Engineering (CSE)

School of Sciences and Engineering

Semester: (Summer, Year: 2025)

PROJECT REPORT

Course Title: Introduction to Programming Language II (Java) Lab
Course Code: CSE282.2
Batch: 65

Project Title: Developing a system for Bank Management in JAVA applying all the pillars of Object-Oriented Programming.

Submitted to:

Dr. Mohammed Ashikur Rahman
Associate Professor
Department of Computer Science and Engineering
Southeast University

Student Details

	Name	ID
1.	Jubair Abdullah	2023200000773
2.	Muhammad Faujul Kabir	2023200000794
3.	Md. Mahmud Hossain	2023200000799

Date of Submission: **-09-25

Table of contents

01.Abstract

This project presents the design and implementation of a Bank Account Management System in Java using Object-Oriented Programming (OOP) principles. The system supports essential banking operations such as account creation, deposits, withdrawals, transfers, and transaction history. The project demonstrates the four pillars of OOP: abstraction, encapsulation, inheritance, and polymorphism. The implementation is console-based, uses in-memory storage, and emphasizes correctness, simplicity, and clarity.

02. Introduction

Banking systems are among the most widely used software applications in the world. Every day, millions of people interact with banking systems for transactions such as deposits, withdrawals, and transfers. Traditionally, banks relied on manual methods to maintain records, which were prone to errors, inefficiency, and data loss.

The goal of this project is to design and implement a Bank Account Management System that demonstrates the principles of OOP in solving real-world problems. Using Java, we created a program that can handle multiple types of accounts, enforce business rules, and log transactions.

The project not only strengthens technical knowledge in Java programming but also enhances problem-solving skills, debugging experience, and teamwork

03. Problem Statement

In manual banking systems, account management is slow and error-prone. Customers face difficulties in tracking their balances, while admins struggle with account maintenance.

Problems with manual systems:

- Delays in processing transactions.
- Higher chances of human error.
- Difficulties in maintaining transaction history.
- No automation in enforcing rules like minimum balance or overdraft limits.

Solution with our system:

- Automated deposit, withdrawal, and transfer operations.
- Separate menus for Admins and Customers.
- Enforced business rules depending on account type.
- Transaction history with timestamps.
- User-friendly console-based interface.

04. Understanding the requirements of the task

Functional Requirements:

- Admin can create, delete, and list accounts.

- Customer can deposit, withdraw, transfer money, check balance, and print history.

Non-Functional Requirements:

- Console-based interaction.

- Error-handled input.

- Fast execution with in-memory storage.

Account Rules:

- Savings Account: Minimum balance 500.

- Current Account: Overdraft up to 1000.

- Student Account: Withdrawal limit 5000 per transaction.

Future Enhancements:

- Persistent storage using a database.
- Graphical User Interface (GUI).
- Multi-user authentication.
- Security features such as password encryption.

05. Analyzing Requirements and Develop the algorithms

Input-Process-Output Model

- **Input:** Menu choices, account type, amounts, account numbers.
- **Process:** Perform transactions according to rules.
- **Output:** Updated balances, success/failure messages, transaction logs.

Algorithms

5.1 Main Menu Algorithm

1. Start program.
2. Display:
    - 1 → Admin Login

- 2 → Customer Portal
- 0 → Exit

3. Read choice.
4. If 1 → go to Admin Menu.
5. If 2 → go to Customer Menu.
6. If 0 → Exit program.
7. Else → show error and repeat.

5.2 Admin Menu Algorithm

1. Verify admin credentials.
2. If valid, show:
   - 1 → Create Account
   - 2 → Delete Account
   - 3 → List Accounts
   - 0 → Back
3. Read choice.
4. Execute operation accordingly.
5. Repeat until exit.

5.3 Create Account Algorithm

1. Input customer name.
2. Input account type.
3. Input opening balance.
4. Generate account number.
5. Create account object.
6. Save account in records.
7. Print success message.

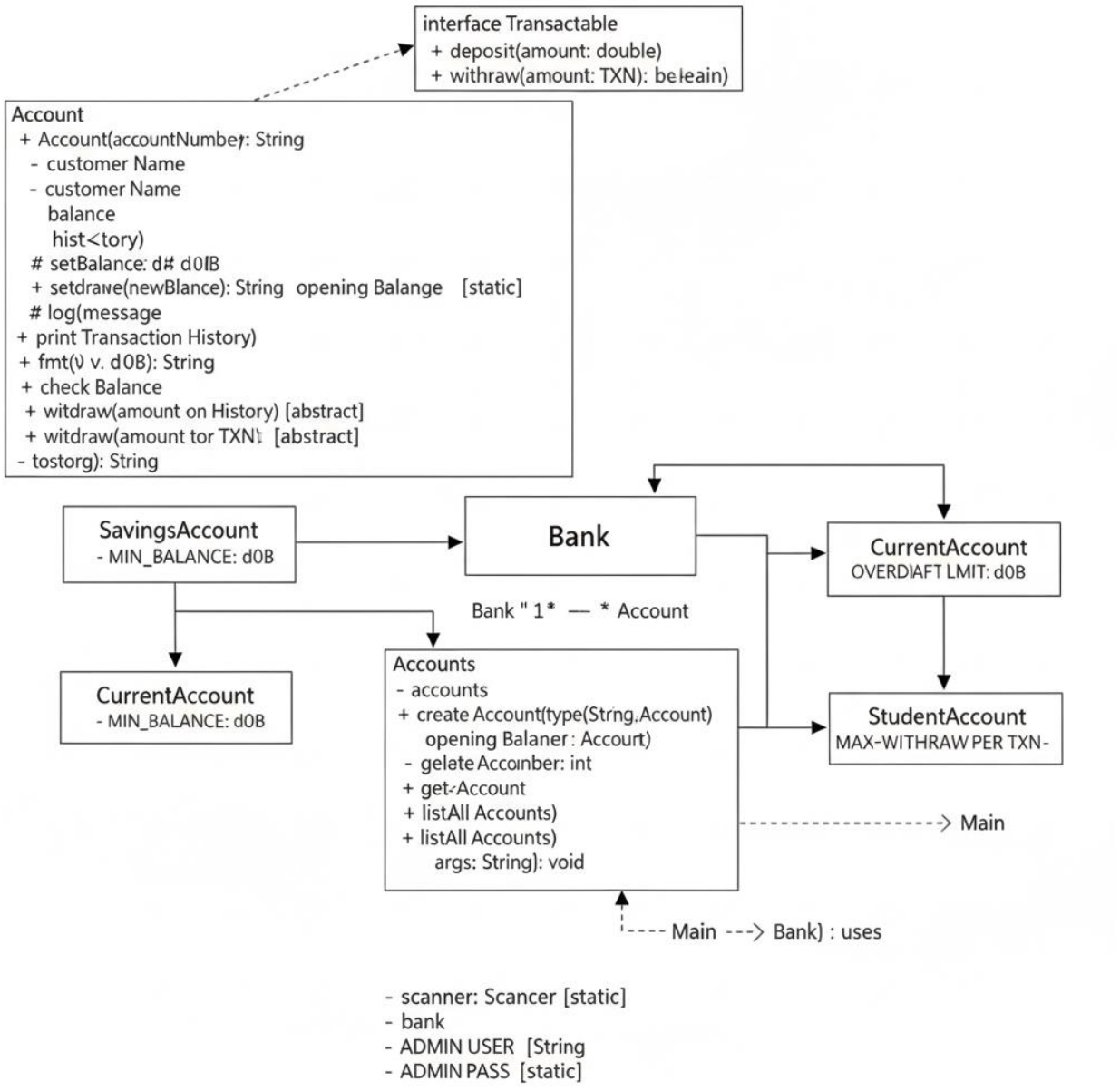
5.4 Withdraw Algorithm

1. Input account number and withdrawal amount.
2. Identify account type.
3. Apply rules:
   - Savings → maintain minimum balance.
   - Current → check overdraft.
   - Student → check per-transaction limit and non-negative balance.
4. If valid → deduct and log.
5. Else → reject with message.

5.5 Transfer Algorithm

1. Input source and destination account numbers.
2. Input transfer amount.
3. Validate both accounts.
4. Withdraw from source (check rules).
5. If success → deposit to destination.
6. Log transfer on both accounts.
7. Print success message.

UML Class Diagram



06. Use of Modern Tools

- **Programming Language:** Java (JDK 24).
- **IDE Used:** Apache NetBeans 25.
- **Libraries:** java.util for collections, java.time for timestamps.
- **Testing & Debugging:** Console-based testing, IDE debugger.
- **Version Control:** GitHub repository for backup and tracking.

07. Application of OOP Concepts

Abstraction

- Implemented via Account (abstract class) and Transactable (interface).
- Abstract withdraw() method allows each account type to define unique rules.

Encapsulation

- Private fields (balance, accountNumber) protected using getters and setters.
- Direct access to sensitive data (like balance) is restricted.

Inheritance

- SavingsAccount, CurrentAccount, and StudentAccount extend Account.
- They inherit properties (account number, customer name) and common methods (deposit, transaction history).

Polymorphism

- Each subclass overrides withdraw() to implement unique withdrawal policies.
- This allows different behaviors while sharing the same interface.

08. Results and Discussion

The system successfully meets all requirements. Each account type enforces its business rules, and transactions are logged properly. The project demonstrates OOP concepts clearly and provides a foundation for real-world systems.

Strengths:

- Clear structure and modular design.
- Demonstrates all four OOP pillars.
- Easy to extend and maintain.

Challenges:

- Designing different rules per account.
- Handling invalid input gracefully.
- Ensuring encapsulation of data.

09. Conclusion

The project achieved its goal of demonstrating OOP concepts in Java through a banking system. All features such as account creation, deposit, withdrawal, transfer, and history were implemented successfully.

Skills learned:

- Java programming with OOP.
- Debugging and testing techniques.
- Designing algorithms and flowcharts.

Future improvements:

- Persistent data storage with databases.
- GUI for better usability.
- Enhanced security for accounts.

10. References

a) Oracle Java Documentation: https://docs.oracle.com/javase/
b) Course lecture slides on OOP and Java.
c) GeeksforGeeks Java Tutorials.

11. Appendices

Full program code:

```
import java.util.*;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;



/**

 * Bank Management System (Console)

 * ---------------------------------

 * Improved version with:

 *  - checkBalance() method

 *  - Better Admin "Show All Accounts" formatting
```

```
 *  - Removed automatic demo accounts
 *  - Clearer output after transactions
 */
public class BankManagementSystem {

    /* ==========================
     * ======  INTERFACES  ======
     * ==========================
     */

    interface Transactable {
        void deposit(double amount);
        boolean withdraw(double amount);
    }

    /* ======================
     * ======  MODELS  =======
     * ======================
     */

    static abstract class Account implements Transactable {
        private final String accountNumber;
        private final String customerName;
        private double balance;
        private final List<String> history;

        protected Account(String accountNumber, String customerName, double openingBalance) {
            this.accountNumber = accountNumber;
```

```
        this.customerName = customerName;
        this.balance = openingBalance;
        this.history = new ArrayList<>();
        log("ACCOUNT CREATED with opening balance: " + fmt(openingBalance));
    }

    public String getAccountNumber() { return accountNumber; }
    public String getCustomerName() { return customerName; }
    public double getBalance() { return balance; }

    protected void setBalance(double newBalance) {
        this.balance = newBalance;
    }

    protected void log(String message) {
        String time = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
        history.add("[" + time + "] " + message);
    }

    public void printTransactionHistory() {
        System.out.println("\n===== Transaction History for " + accountNumber + " (" + customerName + ")
=====");
        if (history.isEmpty()) {
            System.out.println("No transactions yet.");
        } else {
            for (String h : history) System.out.println(h);
        }
        System.out.println("Current Balance: " + fmt(balance));
```

```
System.out.println("============================================================\n");
    }

    public void checkBalance() {
        System.out.println("Account: " + accountNumber +
                           " | Holder: " + customerName +
                           " | Balance: " + fmt(balance));
    }

    @Override
    public void deposit(double amount) {
        if (amount <= 0) {
            System.out.println("Deposit amount must be positive.");
            return;
        }
        setBalance(getBalance() + amount);
        log("DEPOSIT: +" + fmt(amount) + " | Balance: " + fmt(getBalance()));
        System.out.println("Deposit successful.");
        checkBalance();
    }

    @Override
    public abstract boolean withdraw(double amount);

    @Override
    public String toString() {
        return String.format("%s | %s | %s", accountNumber, padRight(customerName, 18), fmt(getBalance()));
```

```
    }

    protected static String fmt(double v) {
        return String.format(Locale.US, "%.2f", v);
    }

    protected static String padRight(String s, int n) {
        if (s.length() >= n) return s.substring(0, n);
        return s + " ".repeat(n - s.length());
    }
}

static class SavingsAccount extends Account {
    private static final double MIN_BALANCE = 500.00;

    public SavingsAccount(String accountNumber, String customerName, double openingBalance) {
        super(accountNumber, customerName, openingBalance);
    }

    @Override
    public boolean withdraw(double amount) {
        if (amount <= 0) {
            System.out.println("Withdrawal amount must be positive.");
            return false;
        }
        double newBalance = getBalance() - amount;
        if (newBalance < MIN_BALANCE) {
```

```
            System.out.println("Withdrawal denied. Savings must maintain minimum balance of " +
fmt(MIN_BALANCE));
            return false;
        }
        setBalance(newBalance);
        log("WITHDRAW: -" + fmt(amount) + " | Balance: " + fmt(getBalance()));
        System.out.println("Withdrawal successful.");
        checkBalance();
        return true;
    }
  }

  static class CurrentAccount extends Account {
    private static final double OVERDRAFT_LIMIT = 1000.00;

    public CurrentAccount(String accountNumber, String customerName, double openingBalance) {
        super(accountNumber, customerName, openingBalance);
    }

    @Override
    public boolean withdraw(double amount) {
        if (amount <= 0) {
            System.out.println("Withdrawal amount must be positive.");
            return false;
        }
        double newBalance = getBalance() - amount;
        if (newBalance < -OVERDRAFT_LIMIT) {
            System.out.println("Withdrawal denied. Overdraft limit exceeded (" + fmt(OVERDRAFT_LIMIT) + ").");
```

```
            return false;
        }
        setBalance(newBalance);
        log("WITHDRAW: -" + fmt(amount) + " | Balance: " + fmt(getBalance()));
        System.out.println("Withdrawal successful.");
        checkBalance();
        return true;
    }
}

static class StudentAccount extends Account {
    private static final double MAX_WITHDRAW_PER_TXN = 5000.00;

    public StudentAccount(String accountNumber, String customerName, double openingBalance) {
        super(accountNumber, customerName, openingBalance);
    }

    @Override
    public boolean withdraw(double amount) {
        if (amount <= 0) {
            System.out.println("Withdrawal amount must be positive.");
            return false;
        }
        if (amount > MAX_WITHDRAW_PER_TXN) {
            System.out.println("Withdrawal denied. Student per-transaction limit is " +
fmt(MAX_WITHDRAW_PER_TXN));
            return false;
        }
```

```
        double newBalance = getBalance() - amount;
        if (newBalance < 0) {
            System.out.println("Withdrawal denied. Student account cannot go negative.");
            return false;
        }
        setBalance(newBalance);
        log("WITHDRAW: -" + fmt(amount) + " | Balance: " + fmt(getBalance()));
        System.out.println("Withdrawal successful.");
        checkBalance();
        return true;
    }
}

static class Bank {
    private final Map<String, Account> accounts = new HashMap<>();
    private int nextAccountNumber = 100100;

    private String generateAccountNumber() {
        return String.valueOf(nextAccountNumber++);
    }

    public Account createAccount(String type, String customerName, double openingBalance) {
        String accNo = generateAccountNumber();
        Account acc;
        switch (type.toLowerCase(Locale.ROOT)) {
            case "savings":
                acc = new SavingsAccount(accNo, customerName, openingBalance);
                break;
```

```
        case "current":
            acc = new CurrentAccount(accNo, customerName, openingBalance);
            break;
        case "student":
            acc = new StudentAccount(accNo, customerName, openingBalance);
            break;
        default:
            throw new IllegalArgumentException("Unknown account type: " + type);
    }
    accounts.put(accNo, acc);
    return acc;
}

public boolean deleteAccount(String accountNumber) {
    return accounts.remove(accountNumber) != null;
}

public Account getAccount(String accountNumber) {
    return accounts.get(accountNumber);
}

public void listAllAccounts() {
    if (accounts.isEmpty()) {
        System.out.println("No accounts available.");
        return;
    }
    System.out.println("\n================= ALL ACCOUNTS =================");
    System.out.println("Acc No   | Customer Name       | Account Type | Balance");
```

```
    System.out.println("-----------------------------------------------------");
    for (Account a : accounts.values()) {
        String type = a.getClass().getSimpleName();
        System.out.printf("%-8s | %-20s | %-12s | %s\n",
                a.getAccountNumber(), a.getCustomerName(), type, Account.fmt(a.getBalance()));
    }
    System.out.println("=====================================================\n");
}

public boolean transfer(String fromAccNo, String toAccNo, double amount) {
    if (amount <= 0) {
        System.out.println("Transfer amount must be positive.");
        return false;
    }
    Account from = getAccount(fromAccNo);
    Account to = getAccount(toAccNo);
    if (from == null || to == null) {
        System.out.println("One or both accounts do not exist.");
        return false;
    }
    if (!from.withdraw(amount)) {
        System.out.println("Transfer failed: could not withdraw from source account.");
        return false;
    }
    to.deposit(amount);
    from.log("TRANSFER OUT -> " + to.getAccountNumber() + " : -" + Account.fmt(amount));
    to.log("TRANSFER IN  <- " + from.getAccountNumber() + " : +" + Account.fmt(amount));
    System.out.println("Transfer successful.");
```

```
        from.checkBalance();
        to.checkBalance();
        return true;
    }
}

private static final Scanner scanner = new Scanner(System.in);
private static final Bank bank = new Bank();
private static final String ADMIN_USER = "admin";
private static final String ADMIN_PASS = "admin123";

public static void main(String[] args) {
    System.out.println("\n===== Welcome to the Bank Management System =====\n");
    mainMenu();
    System.out.println("\nThank you for using the Bank Management System. Goodbye!\n");
}

private static void mainMenu() {
    while (true) {
        System.out.println("Main Menu:");
        System.out.println("1. Admin Login");
        System.out.println("2. Customer Portal");
        System.out.println("0. Exit");
        System.out.print("Choose an option: ");
        String choice = scanner.nextLine().trim();
        switch (choice) {
            case "1":
                if (adminLogin()) adminMenu();
```

```
                break;
            case "2":
                customerMenu();
                break;
            case "0":
                return;
            default:
                System.out.println("Invalid choice. Try again.\n");
        }
    }
}

private static boolean adminLogin() {
    System.out.print("Enter admin username: ");
    String u = scanner.nextLine().trim();
    System.out.print("Enter admin password: ");
    String p = scanner.nextLine().trim();
    if (ADMIN_USER.equals(u) && ADMIN_PASS.equals(p)) {
        System.out.println("\nAdmin login successful.\n");
        return true;
    }
    System.out.println("Invalid admin credentials.\n");
    return false;
}

private static void adminMenu() {
    while (true) {
        System.out.println("Admin Menu:");
```

```
        System.out.println("1. Create Account");
        System.out.println("2. Delete Account");
        System.out.println("3. List All Accounts");
        System.out.println("0. Back to Main Menu");
        System.out.print("Choose an option: ");
        String choice = scanner.nextLine().trim();
        switch (choice) {
            case "1":
                createAccountFlow();
                break;
            case "2":
                deleteAccountFlow();
                break;
            case "3":
                bank.listAllAccounts();
                break;
            case "0":
                System.out.println();
                return;
            default:
                System.out.println("Invalid choice. Try again.\n");
        }
    }
}

private static void createAccountFlow() {
    System.out.println("\nCreate Account");
    System.out.print("Enter customer name: ");
```

```
    String name = scanner.nextLine().trim();
    System.out.print("Enter account type (savings/current/student): ");
    String type = scanner.nextLine().trim().toLowerCase(Locale.ROOT);
    System.out.print("Enter opening balance: ");
    double opening = readDoubleSafe();
    try {
        Account acc = bank.createAccount(type, name, opening);
        System.out.println("Account created successfully!");
        acc.checkBalance();
    } catch (Exception ex) {
        System.out.println("Failed to create account: " + ex.getMessage());
    }
    System.out.println();
}

private static void deleteAccountFlow() {
    System.out.print("\nEnter account number to delete: ");
    String accNo = scanner.nextLine().trim();
    boolean ok = bank.deleteAccount(accNo);
    System.out.println(ok ? "Account deleted successfully.\n" : "Account not found.\n");
}

private static void customerMenu() {
    while (true) {
        System.out.println("Customer Menu:");
        System.out.println("1. Deposit");
        System.out.println("2. Withdraw");
        System.out.println("3. Transfer");
```

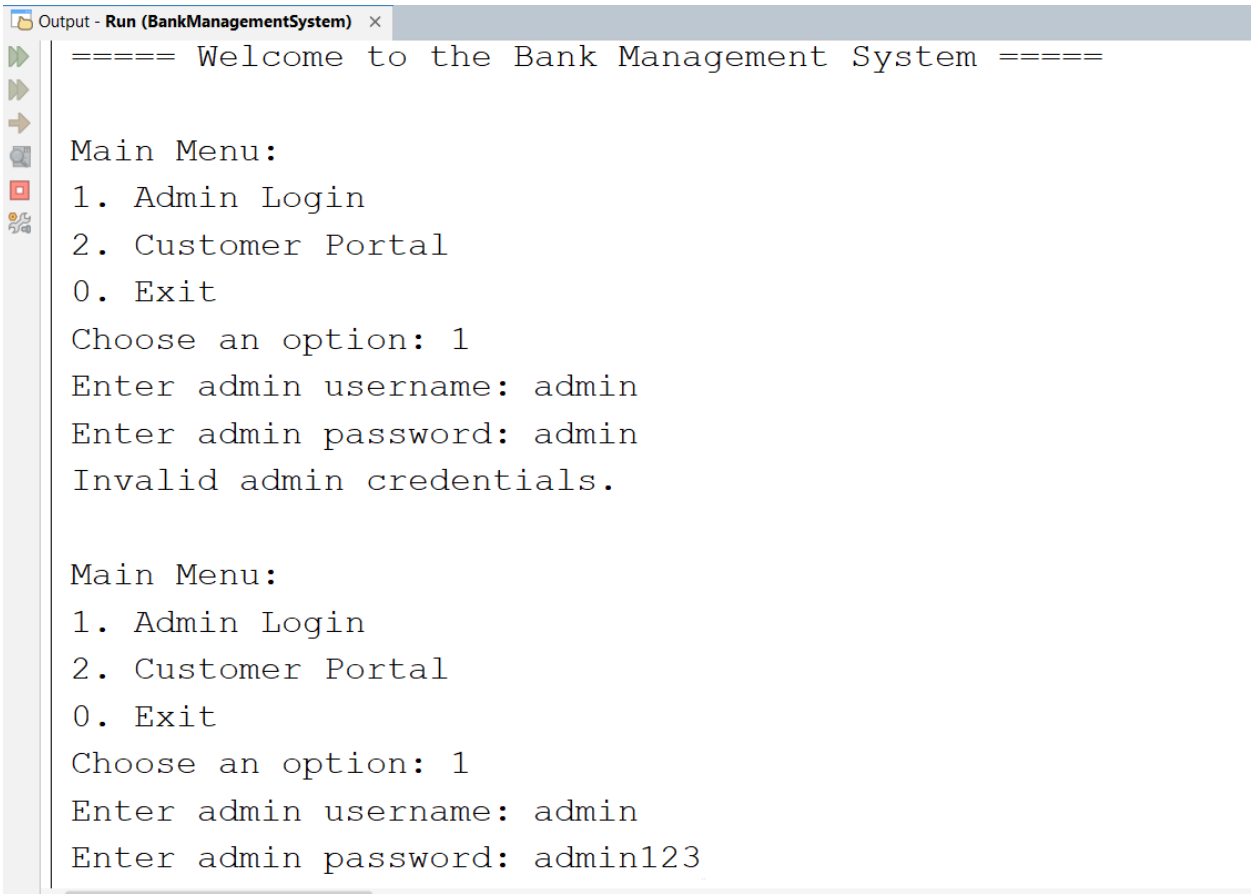
```
        System.out.println("4. Check Balance");
        System.out.println("5. Transaction History");
        System.out.println("0. Back to Main Menu");
        System.out.print("Choose an option: ");
        String choice = scanner.nextLine().trim();
        switch (choice) {
            case "1":
                depositFlow();
                break;
            case "2":
                withdrawFlow();
                break;
            case "3":
                transferFlow();
                break;
            case "4":
                balanceFlow();
                break;
            case "5":
                historyFlow();
                break;
            case "0":
                System.out.println();
                return;
            default:
                System.out.println("Invalid choice. Try again.\n");
        }
    }
```

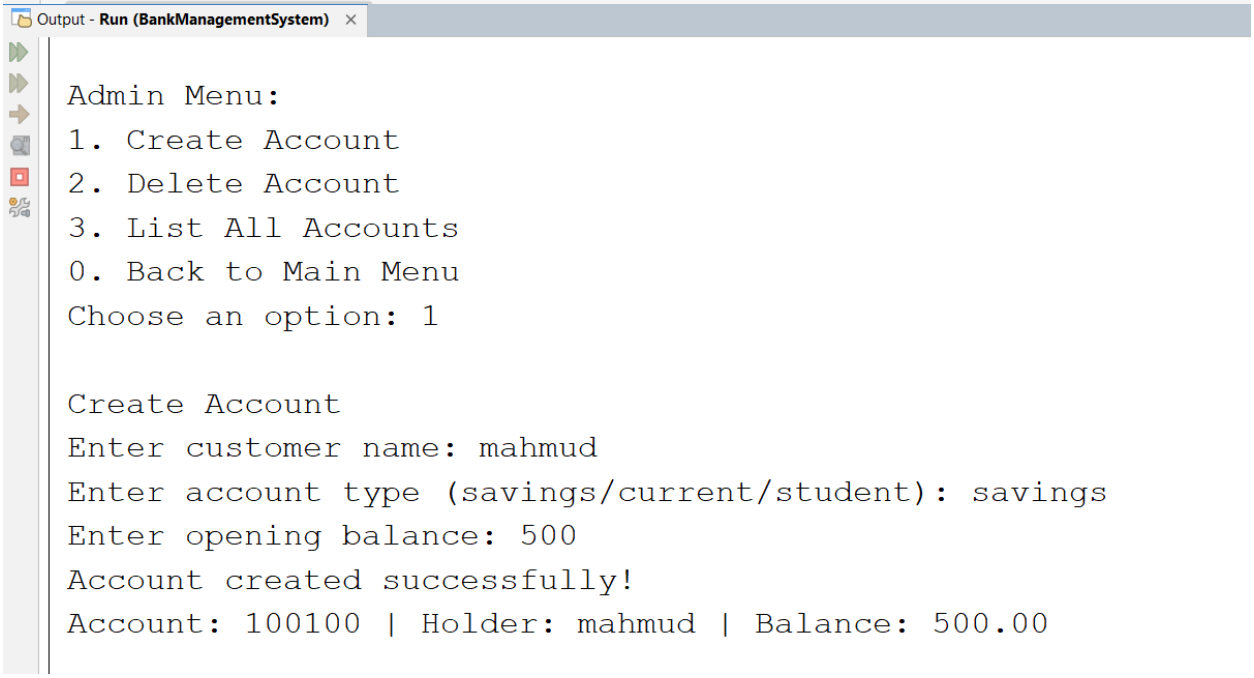
```
}

private static void depositFlow() {
    Account acc = requireAccount("Enter your account number: ");
    if (acc == null) return;
    System.out.print("Enter deposit amount: ");
    double amt = readDoubleSafe();
    acc.deposit(amt);
    System.out.println();
}

private static void withdrawFlow() {
    Account acc = requireAccount("Enter your account number: ");
    if (acc == null) return;
    System.out.print("Enter withdrawal amount: ");
    double amt = readDoubleSafe();
    acc.withdraw(amt);
    System.out.println();
}

private static void transferFlow() {
    System.out.print("Enter your (source) account number: ");
    String from = scanner.nextLine().trim();
    if (bank.getAccount(from) == null) {
        System.out.println("Source account not found.\n");
        return;
    }
    System.out.print("Enter destination account number: ");
```

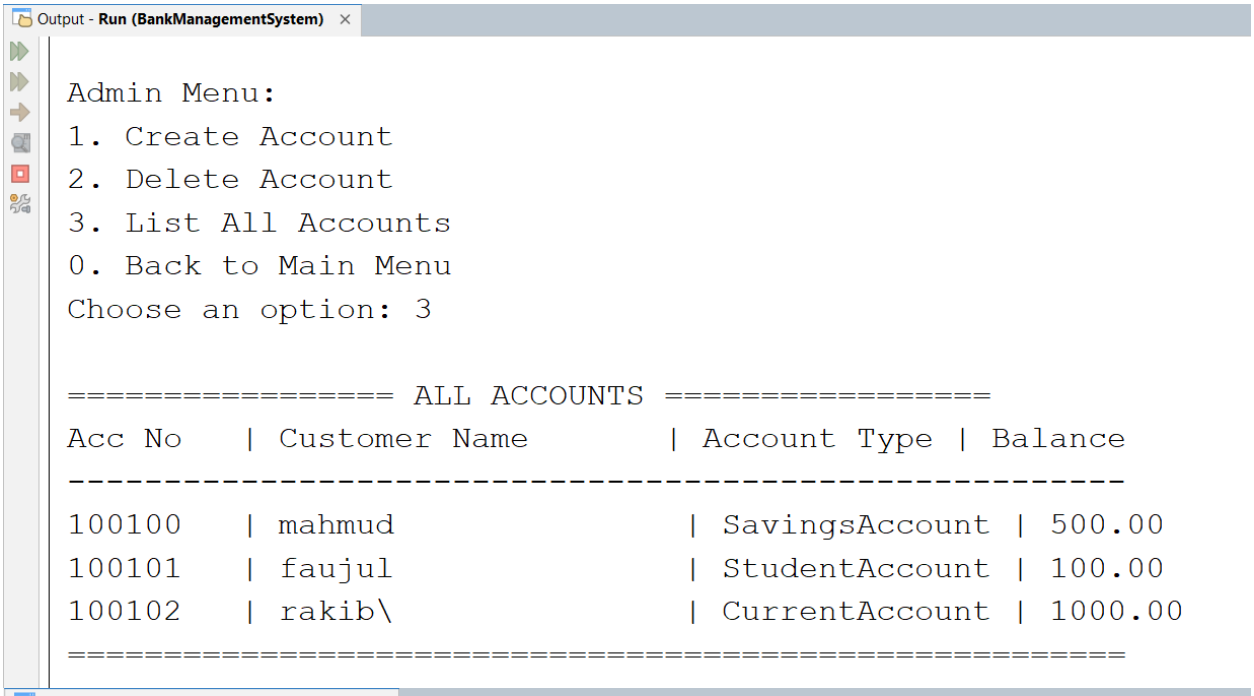
```
    String to = scanner.nextLine().trim();
    System.out.print("Enter transfer amount: ");
    double amt = readDoubleSafe();
    bank.transfer(from, to, amt);
    System.out.println();
}

private static void balanceFlow() {
    Account acc = requireAccount("Enter your account number: ");
    if (acc == null) return;
    acc.checkBalance();
    System.out.println();
}

private static void historyFlow() {
    Account acc = requireAccount("Enter your account number: ");
    if (acc == null) return;
    acc.printTransactionHistory();
}

private static Account requireAccount(String prompt) {
    System.out.print(prompt);
    String accNo = scanner.nextLine().trim();
    Account acc = bank.getAccount(accNo);
    if (acc == null) {
        System.out.println("Account not found.\n");
    }
    return acc;
```

```
    }

    private static double readDoubleSafe() {
        while (true) {
            String s = scanner.nextLine().trim();
            try {
                double v = Double.parseDouble(s);
                return v;
            } catch (NumberFormatException e) {
                System.out.print("Invalid number. Try again: ");
            }
        }
    }
}
```

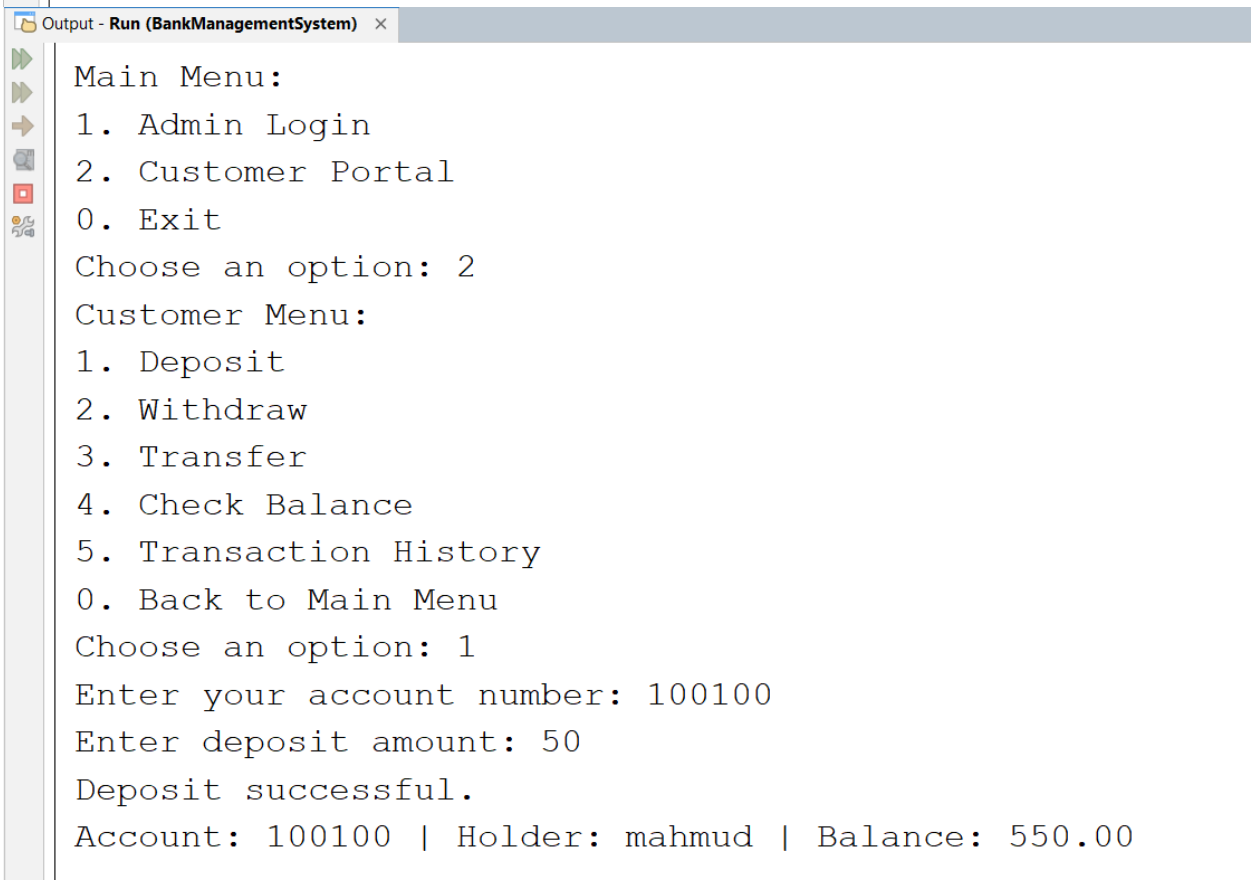
Sample input/output screenshots:



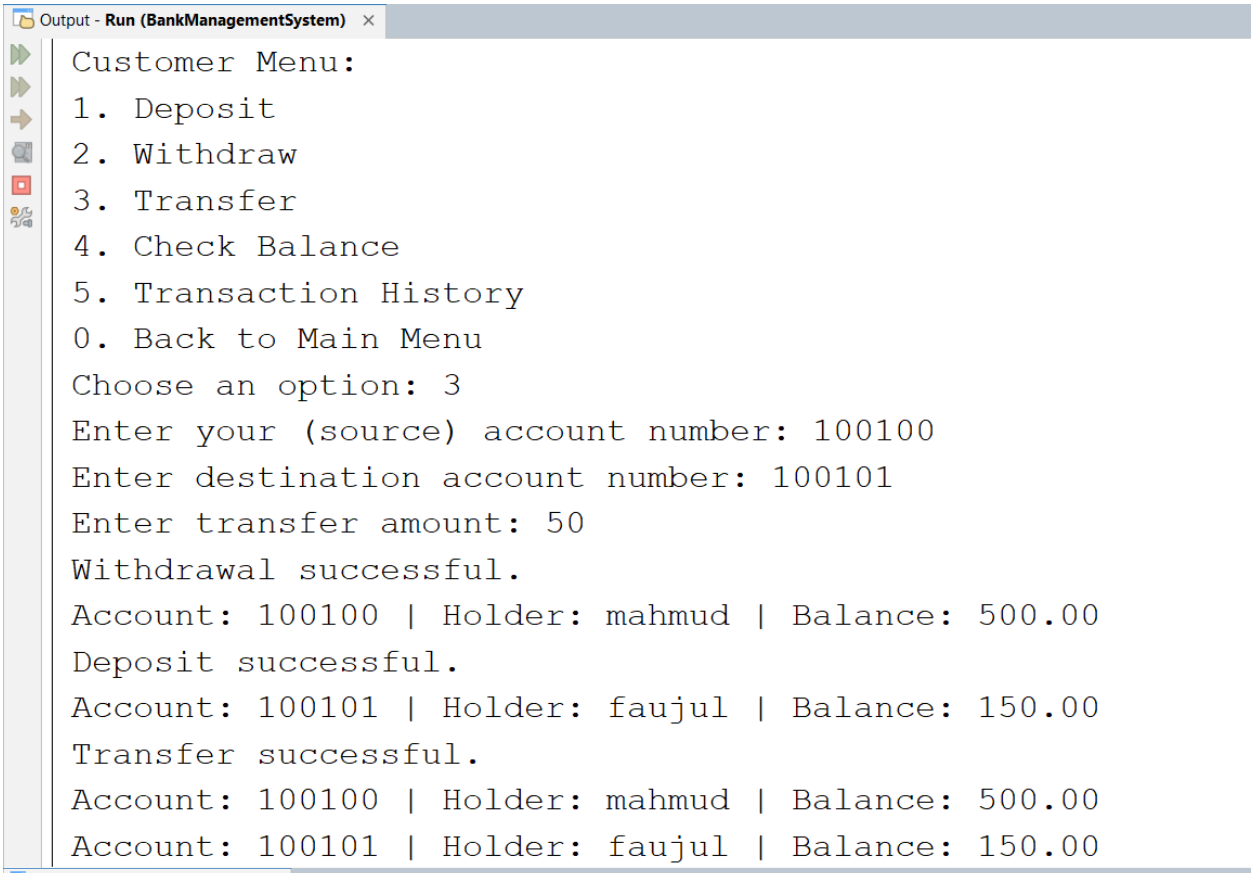


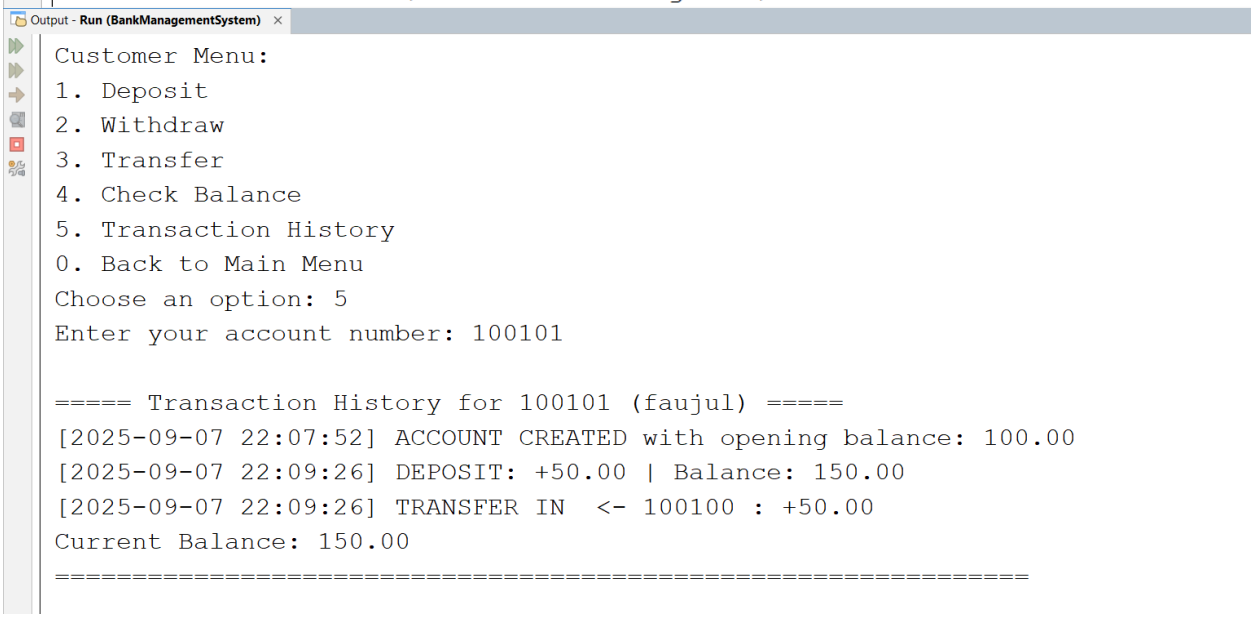


```
Output - Run (BankManagementSystem)

Admin Menu:
1. Create Account
2. Delete Account
3. List All Accounts
0. Back to Main Menu
Choose an option: 3

================= ALL ACCOUNTS =================
Acc No    | Customer Name        | Account Type | Balance
-------------------------------------------------------
100100    | mahmud                | SavingsAccount | 500.00
100101    | faujul                | StudentAccount | 100.00
100102    | rakib\                | CurrentAccount | 1000.00
=======================================================
```



```
Output - Run (BankManagementSystem)

Main Menu:
1. Admin Login
2. Customer Portal
0. Exit
Choose an option: 2
Customer Menu:
1. Deposit
2. Withdraw
3. Transfer
4. Check Balance
5. Transaction History
0. Back to Main Menu
Choose an option: 1
Enter your account number: 100100
Enter deposit amount: 50
Deposit successful.
Account: 100100 | Holder: mahmud | Balance: 550.00
```


Output - Run (BankManagementSystem)
Customer Menu:
1. Deposit
2. Withdraw
3. Transfer
4. Check Balance
5. Transaction History
0. Back to Main Menu
Choose an option: 3
Enter your (source) account number: 100100
Enter destination account number: 100101
Enter transfer amount: 50
Withdrawal successful.
Account: 100100 | Holder: mahmud | Balance: 500.00
Deposit successful.
Account: 100101 | Holder: faujul | Balance: 150.00
Transfer successful.
Account: 100100 | Holder: mahmud | Balance: 500.00
Account: 100101 | Holder: faujul | Balance: 150.00


Output - Run (BankManagementSystem)
Customer Menu:
1. Deposit
2. Withdraw
3. Transfer
4. Check Balance
5. Transaction History
0. Back to Main Menu
Choose an option: 5
Enter your account number: 100101
===== Transaction History for 100101 (faujul) =====
[2025-09-07 22:07:52] ACCOUNT CREATED with opening balance: 100.00
[2025-09-07 22:09:26] DEPOSIT: +50.00 | Balance: 150.00
[2025-09-07 22:09:26] TRANSFER IN <- 100100 : +50.00
Current Balance: 150.00
==========