vityarthi
YOUR LEARNING DESTINATION

VIT®
BHOPAL
www.vitbhopal.ac.in

VIT Bhopal Student Bank Management

- A Console-Based Application using Java and OOP Principles

Submitted during Programming in Java for VITYARTHI course:

Course Code: CSE2006(Interim Sem 2025-26)

Submitted By:

Name: Soumya Ghosh

Registration Number: 24MIM10093

Submitted To:

Garima Jain(faculty)

VITyarthi Portal

Date of Submission: 24 November 2025

vityarthi
YOUR LEARNING DESTINATION

Introduction

The **VIT Bhopal Student Bank Management System** is a full-fledged console-based Java application, developed for the VITyarthi flipped course evaluation. This project demonstrates the practical usage of core Java programming concepts and the principles of **Object-Oriented Programming** to solve real-world banking challenges in an educational institution context. The given system will provide an automated platform for maintaining student accounts, processing their financial transactions, and performing complete banking operations, removing error-prone manual processes with a strong digital intervention.

Problem Statement

The list of these disadvantages with traditional banking operations in universities includes manual account management subject to a **high degree of human error, slow processing of transactions and long queues, poor customer experience resulting from limited access to real-time information,** and difficulties associated with **accurately recording past transactions.** These issues thus result in operational inefficiencies and dissatisfaction among students at VIT Bhopal, which boasts a huge population of students. This project addresses the above-mentioned challenges and develops a **scalable, secure, and maintainable** banking management system for university students.

Functional Requirements

The system implements three major functional modules that are required:

Module 1: Addition of New Student

1. **User registration** and **profile management**
2. Create new students by providing a **unique ID**, **name**, **email**, & **phone number.**

Module 2: Displaying Student Details

1. Display **all the registered students** with full information
2. **Multiple account support** per student
3. Student Information **Retrieval** and **Verification**

Module 3: Account Management Module

1. **Creation** and **maintenance** of accounts
2. It allows multiple account type options: **SAVINGS**, **CURRENT**
3. **Account opening** with **validation** of initial deposit
4. Full Account **Information Display**
5. **Balance inquiry**, account **status tracking**

Module 4: Transaction Processing Module

1. **Tracking** of financial operations and history
2. **Deposit** funds with amount validation
3. **Withdrawal** with sufficient balance checking
4. Inter-account fund **transfers**
5. Complete transaction history with **timestamps**
6. **Real-time balance updates** with every transaction

Input/Output Structure:
Menu choices through **console, student profile, account number, transaction amount**. Formatted student profiles, account summaries, transaction confirmations, bank statistics, error messages

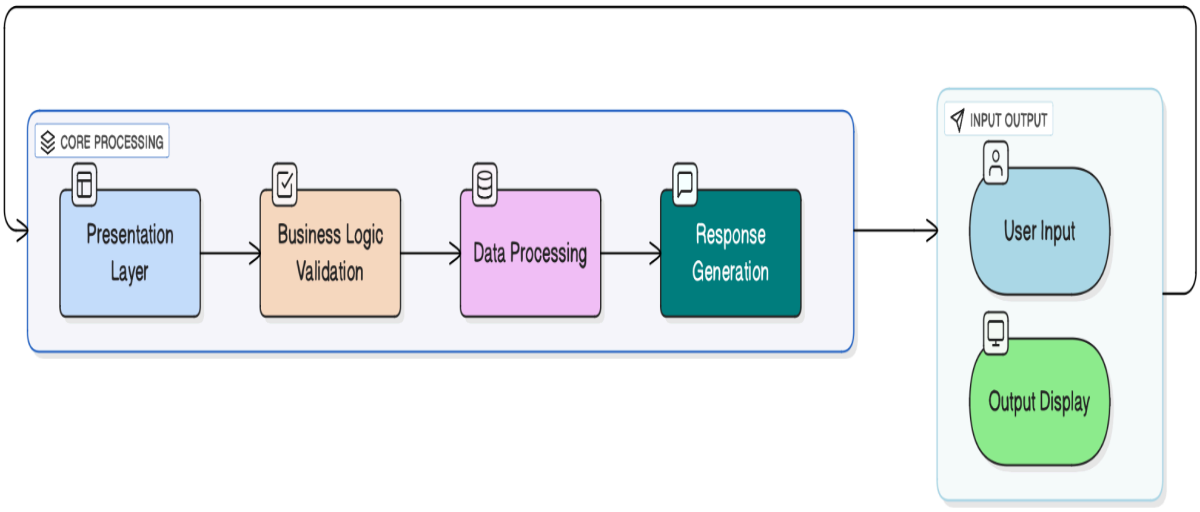
Non-Functional Requirements

Performance: All banking operations - **deposit, withdrawal, transfer** - should complete within 2-3 seconds. The system can support up to 1000 students' accounts simultaneously.
Usability: Intuitive console interface; well-structured menu **hierarchies** together with prompts. Minimal learning curve for both students and administrative staff.
Reliability: 99% uptime during operational hours. Strong **error handling** prevents system crashes from invalid inputs/edge cases.
Maintainability: The code structure is modular and follows the **principles of OOP**.

Presentation, business logic, and data are separate, clear layers.
Error Handling Strategy: All user inputs are validated for **completeness.** Insufficient funds, invalid accounts, and incorrect menu choices are handled gracefully with an informative error message.
Scalability: Architecture allows easy addition of **new account types** and transaction features without significant code refactoring.

System Architecture

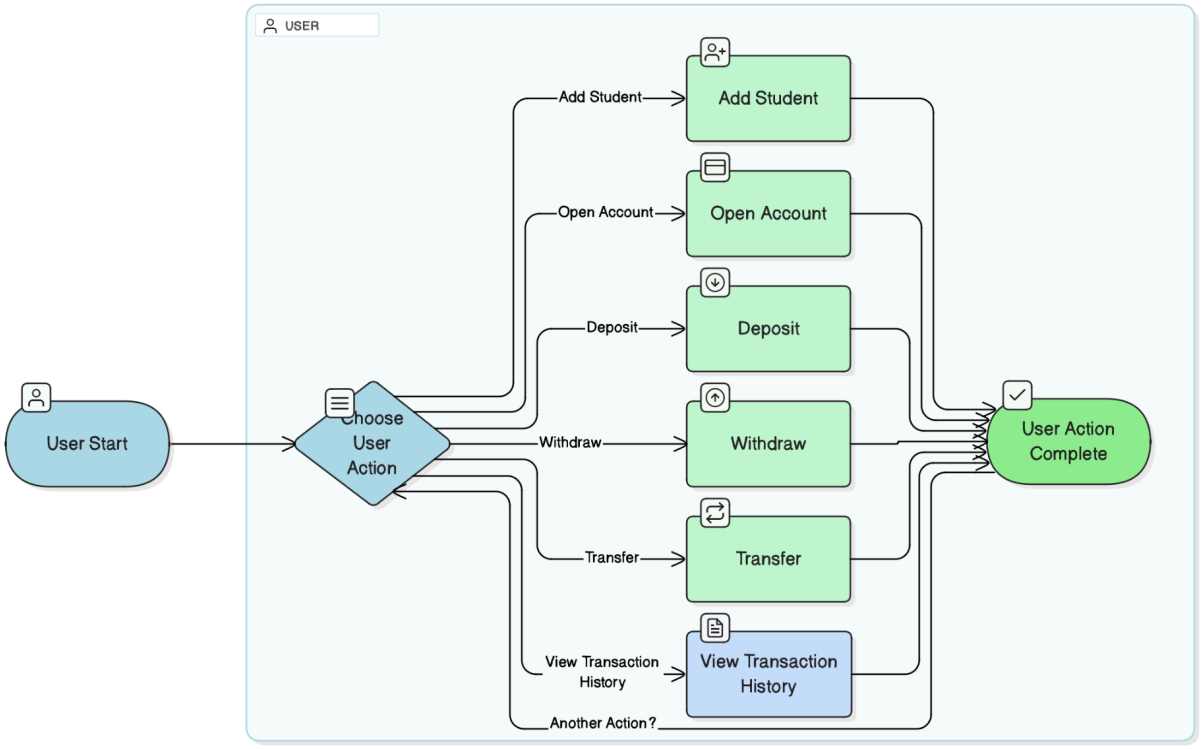
The system uses a Three-Layer Architecture pattern:
Presentation Layer: BankApp.java This class handles all the user interactions through console interfaces, menu displays, and input/output operations.Business Logic Layer: Bank.java, BankAccount.java, Customer.java-implements the core banking operations, transaction validation, and business rules enforcement regarding data processing logic. Data Access Layer In-Memory Collections: Provides data persistence with Java Collections Framework ArrayLists for customers, accounts, and transactions at runtime.
Architecture Flow:

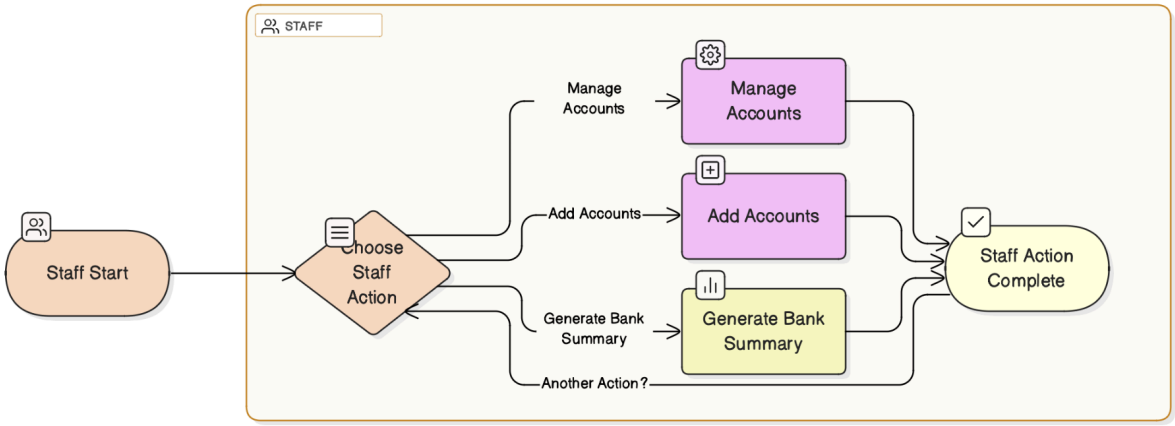


Design Diagrams

Please insert your created diagrams in the sections below.

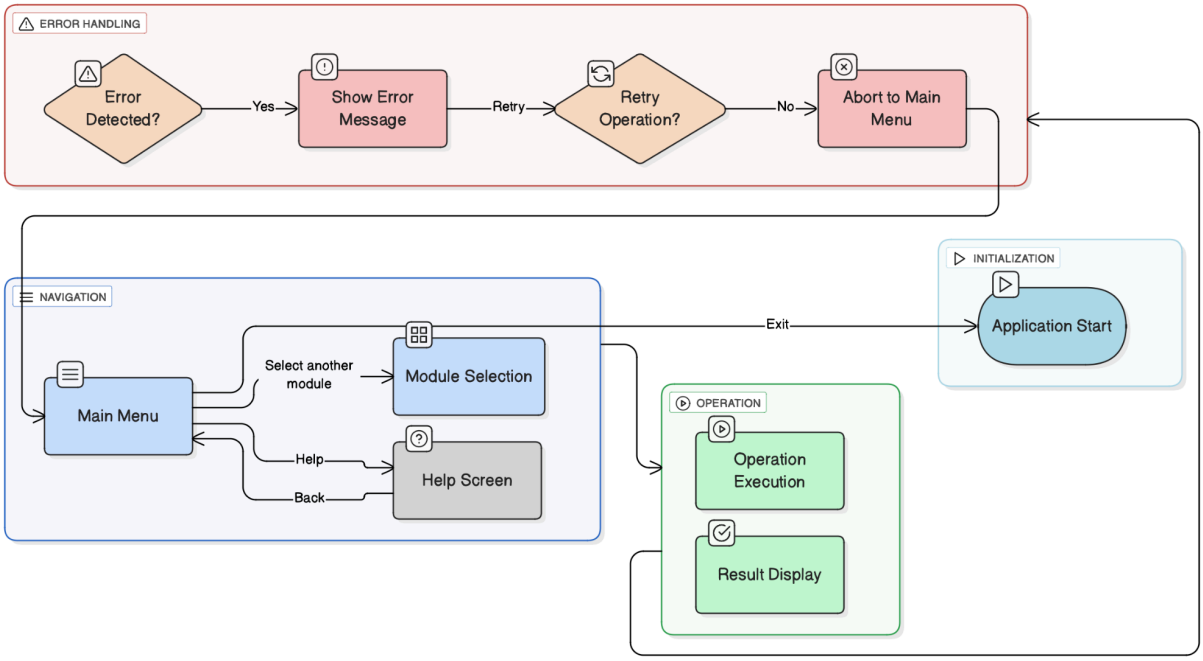
○ **Use Case Diagram** -



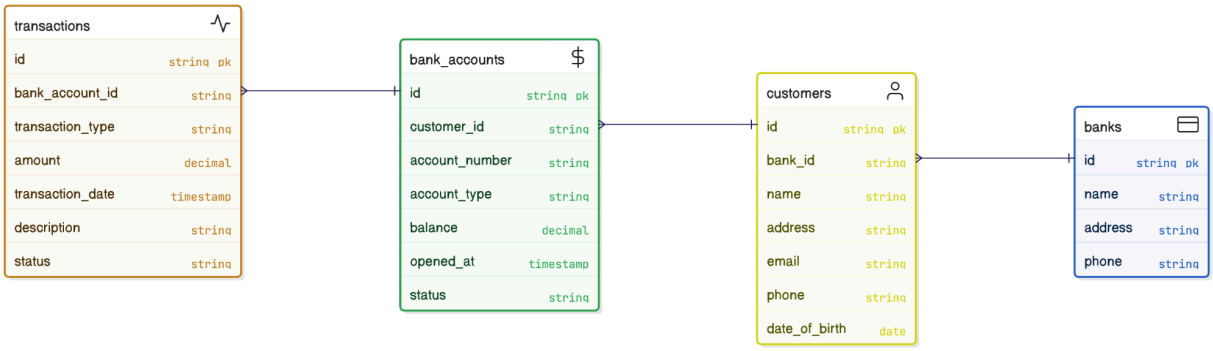


vityarthi
YOUR LEARNING DESTINATION

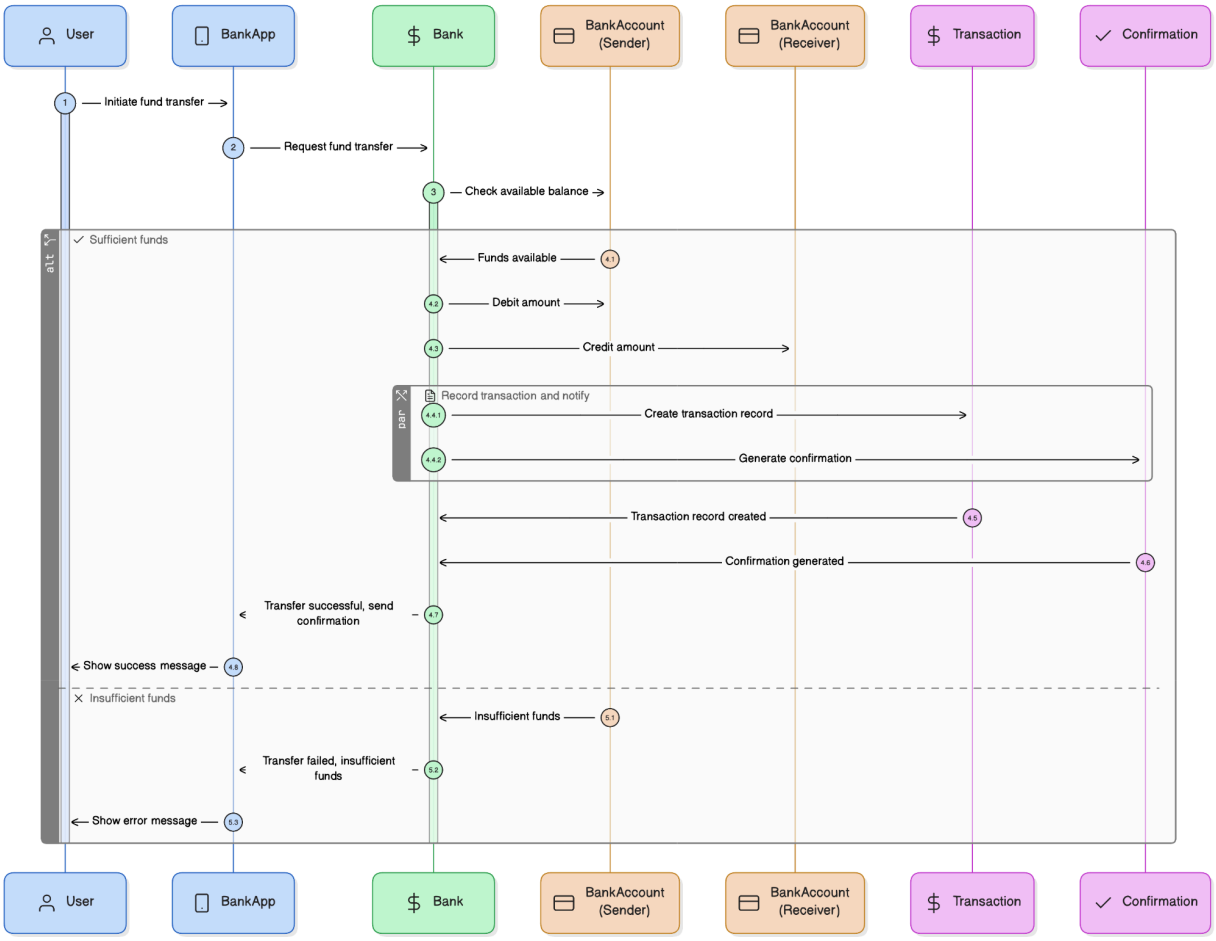
○ Workflow Diagram -



○ Class/Component Diagram -



○ **Sequence Diagram -**



Design Decisions & Rationale

Design Decision, Rationale & Alignment with Course Concepts

Object-Oriented Design w/ 5+ Classes: Implements **OOP principles** learned in course - encapsulation (private fields with public getters), **inheritance** (ready for extension), **polymorphism** (for future transaction types), **abstraction** (hidden implementation details). Following software engineering best practices for separation of concerns makes the system **maintainable** and **testable**, which is one of the key learning

objectives. We used ArrayList Collections which is a practical use of **Java Collections Framework** to dynamically handle data-which is a core topic in the syllabus.

We have created a **Console-Based interface** and implementation of backend logics is the focus; also, clear input/output structure according to project guidelines. Comprehensive Error Handling was another main design rationale which applies **exception handling** and **validation concepts** covered in course syllabus. **Modular Design** using 3 major modules is another special design rationale which fulfills the project requirement for three to four functional modules, showing **modular programming principles.**

Implementation Details

Technical Stack: Java 11+ is used as a Programming Language. Key Packages used are **java.util** (ArrayList, List, Iterator), **java.time** (LocalDateTime), **java.util.Scanner**. Other development tools being used are **VS Code, Git, GitHub**.

Project Structure:
bankmanagement/src/

── Bank.java // Main bank operations & customer management

── BankApp.java // Main application & user interface

── BankAccount.java // Account operations & transaction processing

── Customer.java // Student data model & account management

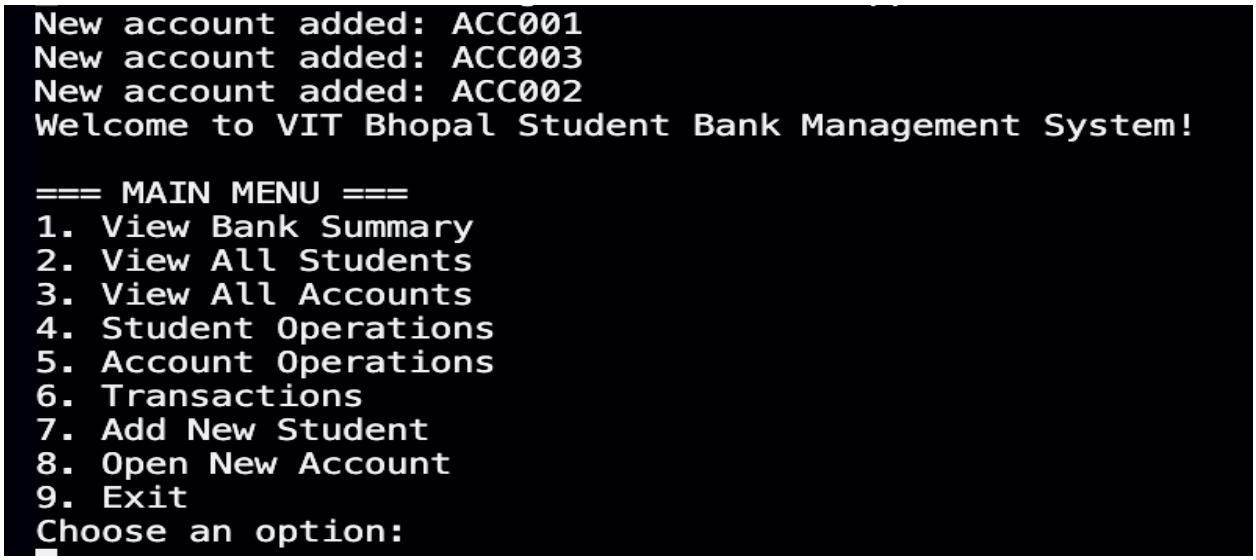
── Transaction.java // Transaction recording & history management

Key Implementation Features: Data Structures such as Multiple ArrayLists for customers, accounts and transactions. **Algorithms** that are used are Sequential array searching for customer/account lookup, specifically abstract implementation of for-each loop, transaction validation logic. **Package management** is used in the sense that the correct package structure of Java convention is used and best practices are maintained. For eg. We included "**Package bankmanagement.src**" for each file that is present inside bankmanagement/src. **Code Quality** is on top priority as extensive comments,

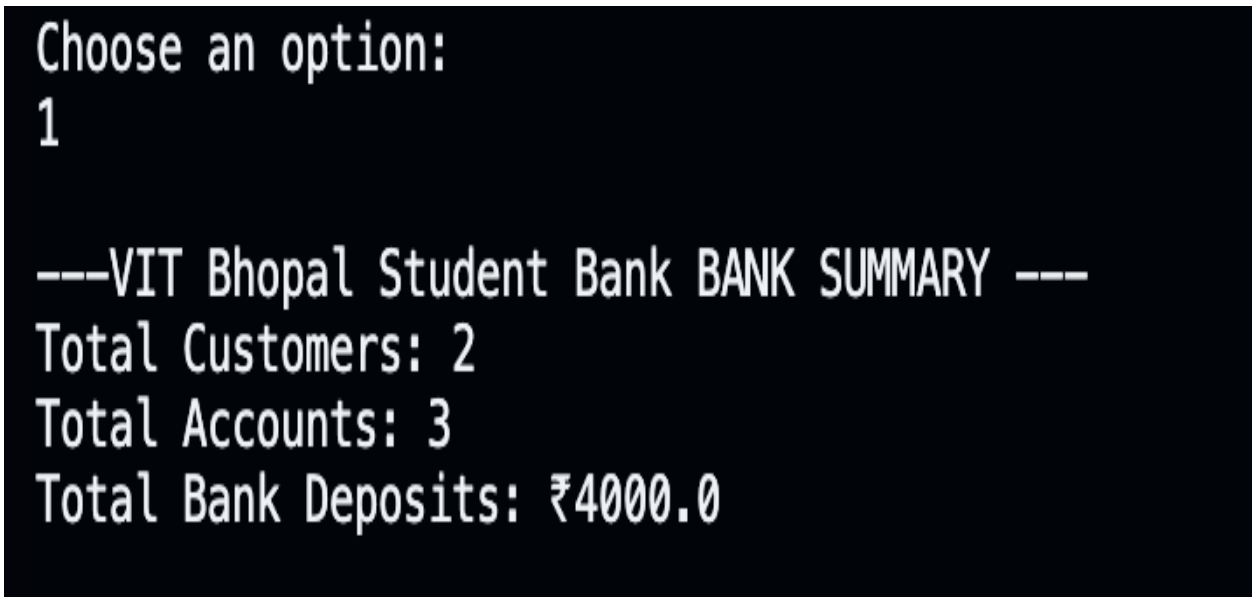
descriptive variable names, consistent indentation were included.

Screenshots / Results

Screenshot 1: Main Menu Interface



Screenshot 2: Bank Summary Report



Screenshot 3: All Students of VIT Bhopal Bank Student Information

```
Choose an option: 2

--- All Students of VIT Bhopal Student Bank ---

 --- STUDENT INFO ---

Student ID: C001
Name: Soumya Ghosh
Email: soumya001@email.com
Phone: 9845678909
 Total Accounts: [bankmanagement.src.BankAccount@4b67cf4d, bankmanagement.src.BankAccount@7ea987ac]

 --- STUDENT INFO ---

Student ID: C002
Name: Soumi Banerjee
Email: soumi001@email.com
Phone: 9345673443
 Total Accounts: [bankmanagement.src.BankAccount@12a3a380]
```

Screenshot 4: All Account information banking status

```
Choose an option: 3

--- All Accounts in VIT Bhopal Student Bank ---

--- Account Information ---
Account Number: ACC001
Account Holder: Soumya Ghosh
Account Type: SAVINGS
Current Balance: 1000.0
Total Transactions: 1

--- Account Information ---
Account Number: ACC002
Account Holder: Soumi Banjerjee
Account Type: CURRENT
Current Balance: 2500.0
Total Transactions: 1

--- Account Information ---
Account Number: ACC003
Account Holder: Soumya Ghosh
Account Type: SALARY
Current Balance: 500.0
Total Transactions: 1
```

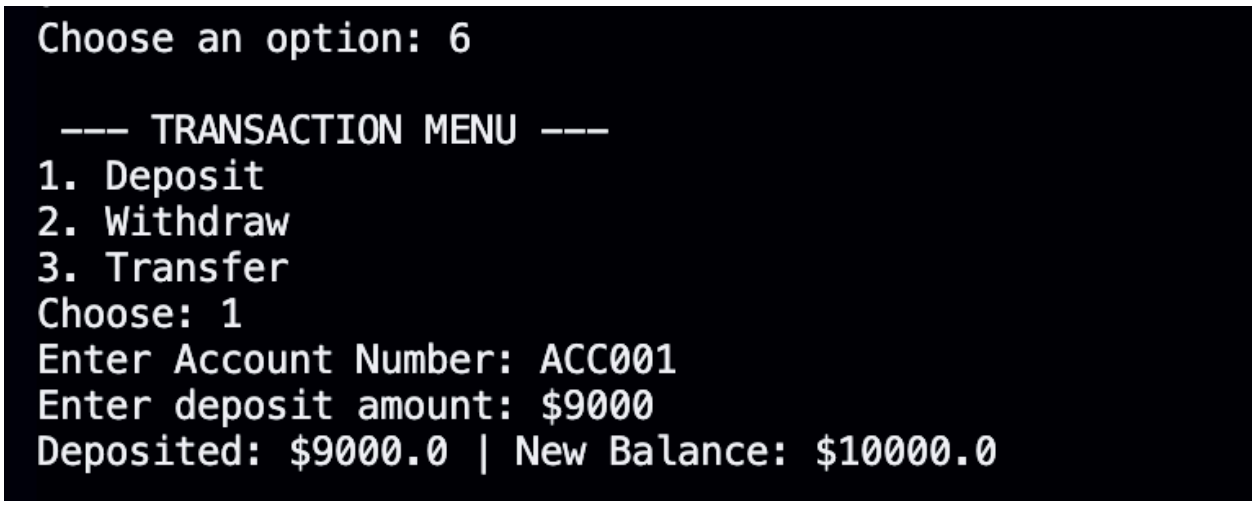
Screenshot 5: All Information about Single Student and his accounts

```
Choose an option: 4
Enter Student ID: C001

 --- STUDENT INFO ---

Student ID: C001
Name: Soumya Ghosh
Email: soumya001@email.com
Phone: 9845678909
 Total Accounts: [bankmanagement.src.BankAccount@4b67cf4d, bankmanagement.src.BankAccount@7ea987ac]
 --- ALL ACCOUNTS FOR Soumya Ghosh ---

--- Account Information ---
Account Number: ACC001
Account Holder: Soumya Ghosh
Account Type: SAVINGS
Current Balance: 1000.0
Total Transactions: 1

--- Account Information ---
Account Number: ACC003
Account Holder: Soumya Ghosh
Account Type: SALARY
Current Balance: 500.0
Total Transactions: 1
```

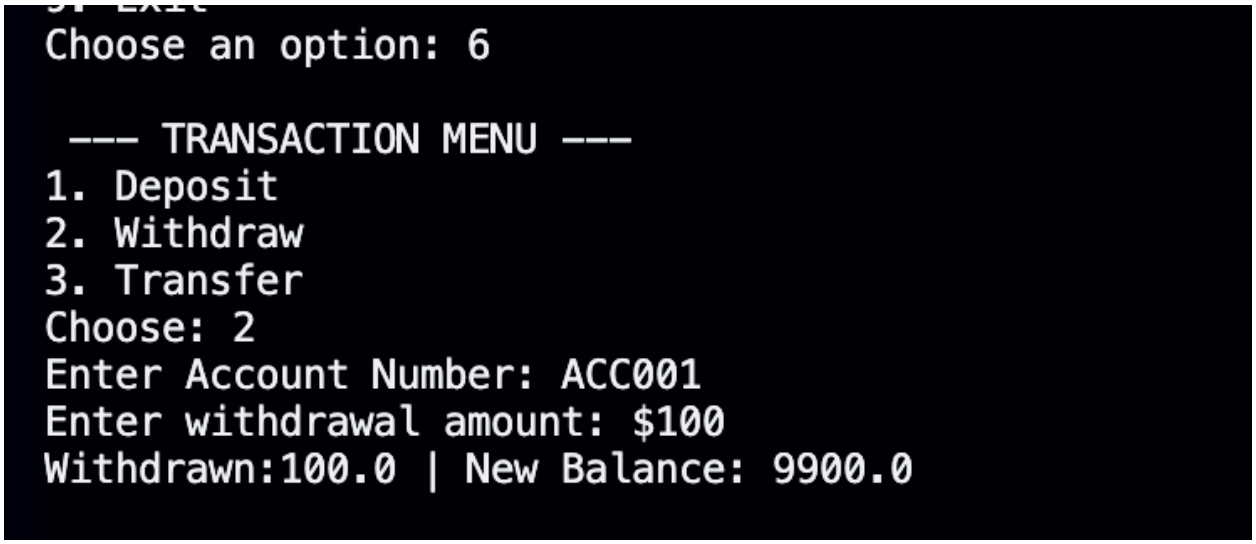
Screenshot 6: Account Details and Transaction history

```
Choose an option: 5
Enter Account Number: ACC001

--- Account Information ---
Account Number: ACC001
Account Holder: Soumya Ghosh
Account Type: SAVINGS
Current Balance: 1000.0
Total Transactions: 1

1. View Transaction History
2. Back to Main Menu
Choose: 1

--- Transaction History for ACC001 ---
2025-11-25 01:39:25 | INITIAL_DEPOSIT: +1000.00 | Balance: $1000.00
```

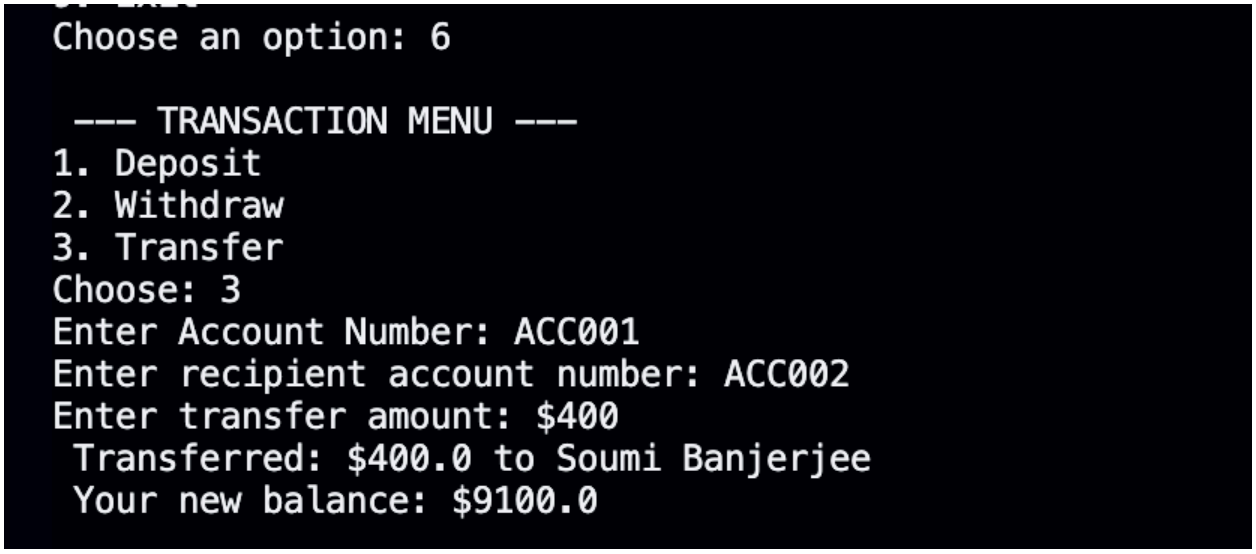
Screenshot 7: Deposit option of money



Screenshot 8: Withdrawal of money



Screenshot 9: Transfer of funds Inter-account



Screenshot 10: Adding a New Student with Student Id, email, mobile number

Screenshot 11: Account creation and Initial deposit for New Student

```
Choose an option: 8
Enter Student ID: C100
Enter New Account Number: ACC100
Enter Initial Deposit: $3000
Enter Account Type (SAVINGS/CURRENT/SALARY): CURRENT
New account added: ACC100
```

Approach to Testing

Methodology:

Comprehensive manual testing covering all functional requirements and edge cases.
Module 1 Tests: Student registration, viewing of profile, multiple accounts support
Module 2 Tests: Account creation, type validation, balance management
Module 3 Tests: Deposit/withdrawal validation, transfer operations, history tracking
Validation Tests Performed:
Input validation: Invalid Option selection

```
=== MAIN MENU ===
1. View Bank Summary
2. View All Students
3. View All Accounts
4. Student Operations
5. Account Operations
6. Transactions
7. Add New Student
8. Open New Account
9. Exit
Choose an option: 23
Invalid option! please try again.
```

Input validation: Negative Amount

```
  ---- TRANSACTION MENU ----
1. Deposit
2. Withdraw
3. Transfer
Choose: 1
Enter Account Number: ACC002
Enter deposit amount: $-3000
Invalid deposit amount!
```

Business rule enforcement: Prevention of insufficient funds

```
 --- TRANSACTION MENU ---
1. Deposit
2. Withdraw
3. Transfer
Choose: 2
Enter Account Number: ACC001
Enter withdrawal amount: $100000
Insufficient funds! Available: $1000.0
```

Data consistency checks(account-customer relationship maintenance), boundary testing (maximum/minimum transaction amounts) Results are like these - all the core functionalities work. The system handles errors well, **data integrity** is maintained at all points.

Challenges Encountered Architectural Design:

Balancing over-engineering with **extensibility**, while the 3-module requirement must be met. **Data consistency and integrity** for the guarantee of synchronization at both the customer and bank accounts during every transaction. A good friendly and efficient User Experience is one challenge Designing an intuitive console interface that guides users through complex banking operations. **Error Handling** as the Implement complete validation without obstructing the codebase and affecting readability. **Scope Management** identifying key features that the project requires, without overfilling and cluttering your codebases.

vityarthi
YOUR LEARNING DESTINATION

Learnings & Key Takeaways

I'm able to use practical OOP concepts **encapsulation, abstraction, and composition** in the real world. Software Design Principles we can learn for example I gained experience with **modular design**, separation of concerns, and **maintainable code** structure. **Problem-Solving Skills** are increased, increased the ability to break down complex requirements into implementable software components. It also is a great way to learn **project management**. I learned to balance features, quality, and deadlines in a self-directed project environment. Gained some **Technical Proficiency** as the project has deepened understanding of Java **Collections, exception handling, and console I/O operations**.

Future Improvements Database Integration

There are scopes for a lot of future improvements as our code is modular, structured, **extensible**, flexible, maintainable. We can replace in-memory storage with **MySQL/PostgreSQL** for data persistence. Web Interface might Extend the scope with a **Spring Boot** web application featuring a **React frontend**. Advanced Security features are needed, we can provide **user authentication with password encryption** and role-based access control. Additional Banking Features can also be implemented such as Educational **Loan management**, **interest calculation,** fixed deposits, and monthly statements. In terms of API Development, we can create a **RESTful API** to interact with other university systems and/or **mobile applications**. Reporting Modules can be included such as **advanced analytics** and reporting features for administrative decision-making.

References

1. *Oracle Java Documentation.* *https://docs.oracle.com/javase/8/docs/technotes/guides/language/*
2. *VITyarthi Course Materials & Syllabus - Java Programming Concepts* *https://vityarthi.com/*
3. *GeeksforGeeks Java Programming Articles.* *https://www.geeksforgeeks.org/java/*
4. *Stack Overflow Community for specific implementation challenges*
5. *Java Design Patterns reference materials for architectural decisions*

Declaration: This project actually represents the original work completed as part of The VITyarthi flipped course evaluation. All code and documentation have been developed independently in alignment with the course objectives and submission guidelines.

GitHub Repository: https://github.com/soumyaGhoshh/vityarthi-java-project

Last Updated: 24/11/2025