Financial Management System Using MVC Architecture

1. Project Overview

The Bank Management System is a Java-based application designed to manage basic banking operations such as account creation, deposit, withdrawal, loan processing, and notifications. The project is built using the **Model-View-Controller (MVC)** architecture to ensure separation of concerns, modularity, and scalability.

2. Objectives

1. To implement a robust and modular banking system.
2. To utilize the MVC architecture for a clear separation of data (Model), user interface (View), and application logic (Controller).
3. To enable functionalities like:
    - Account creation and management (Savings and Current accounts).
    - Transactions (Deposit and Withdrawal).
    - Loan processing.
    - SMS notifications for account activities.

3. Key Features

1. **Account Management**:
    - Support for Savings and Current accounts.
    - Deposit and withdrawal functionality.
    - Interest calculation for savings accounts.
2. **Loan Management**:
    - Loan application and repayment calculation.
3. **Notifications**:
    - SMS notifications for transactions and loan approvals.
4. **User Roles**:
    - Customer and Employee.
5. **Balance Inquiry**:
    - Retrieve and display account balance.

4. MVC Architecture Implementation

The project follows the **MVC** design pattern, which is divided into the following components:

4.1. Model

The **Model** contains the core logic and data of the application:

- **Person (Abstract)**: Base class for Customer and Employee.

- **Customer**: Stores customer-specific data like phoneNumber.
- **Account (Abstract)**: Base class for SavingsAccount and CurrentAccount, encapsulating account details and operations (deposit and withdraw).
- **SavingsAccount**: Implements interest rate calculations.
- **CurrentAccount**: Includes overdraft limit for withdrawals.
- **Loan**: Manages loan details and repayment calculation.
- **SMSNotification**: Sends SMS notifications for account activities (deposits, withdrawals, loan approvals, etc.).

4.2. View

The **View** handles the presentation layer and user interface:

- **BankView** : A simple console-based interface to display information such as customer details, transaction status, and account balance.

4.3. Controller

The **Controller** acts as the intermediary between the Model and the View:

- **BankController**: Manages banking operations like deposits, withdrawals, and notifications. It fetches data from the Model and updates the View.

5. Example Scenarios

5.1. Deposit Scenario

1. The customer requests to deposit money.
2. The BankController calls the deposit() method in the Account model.
3. The deposit() method updates the balance.
4. The SMSNotification class sends a confirmation SMS.
5. The BankView displays a success message.

5.2. Withdrawal Scenario

1. The customer requests to withdraw money.
2. The BankController calls the withdraw() method in the Account model.
3. The withdraw() method checks the balance and deducts the amount.
4. The SMSNotification class sends a confirmation SMS.
5. The BankView displays a success message.

6. Code Structure

Model

- Person (abstract): Base class for Customer and Employee.

- Customer: Stores customer details.
- Account (abstract): Base class for SavingsAccount and CurrentAccount.
- SavingsAccount: Includes interest calculation.
- CurrentAccount: Includes overdraft limit.
- Loan: Manages loan processing.
- SMSNotification: Sends SMS alerts.

View

- BankView: Displays data to the user and captures input.

Controller

- BankController: Manages interactions between the Model and View.

7. Advantages of the Design

1. **Separation of Concerns**:
    - The system is divided into Model, View, and Controller, ensuring each component handles only its responsibilities.
2. **Reusability**:
    - Models like Account and Loan can be reused in other banking systems.
    - Views can be swapped (e.g., replacing the console with a graphical UI) without altering the logic.
3. **Scalability**:
    - New features like additional account types or advanced notifications can be added easily.
4. **Testability**:
    - Each component can be tested independently, ensuring high-quality code.

8. Limitations

1. The console-based interface lacks visual appeal.
2. No persistent data storage (e.g., database integration) is implemented in this version.

9. Future Enhancements

1. **GUI Integration**:
    - Replace ConsoleView with a graphical user interface using JavaFX or Swing.
2. **Database Connectivity**:
    - Store customer, account, and transaction data in a database for persistence.
3. **Advanced Notifications**:
    - Add email notifications and app-based alerts alongside SMS.
4. **Role-Based Access**:
    - Introduce authentication for employees and customers to secure the system.

10. Conclusion

The Financial Management System demonstrates a clear implementation of the **MVC architecture**, providing a modular and scalable solution for basic banking operations. While the console-based version is functional and adheres to best practices, future versions can extend its capabilities with advanced features like GUI and database integration. The project is a good foundation for understanding how to structure real-world applications using object-oriented principles and design patterns.