Project Report on Financial Management System (FMS)

Course Title: Advanced Programming Laboratory
Course Code: 0714 02 CSE 2100

Introduction:

The **Financial Management System (FMS)** is a software application designed to automate and manage the financial operations of an organization or an individual. The system provides capabilities such as account management, transaction processing (deposits and withdrawals), loan management, and notifications. Developed in **Java**, it follows Object-Oriented Programming (OOP) principles such as **Abstraction**, **Encapsulation**, **Inheritance**, and **Polymorphism**, ensuring modular and maintainable code. Additionally, the project adheres to **SOLID principles** (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion) to ensure code quality, scalability, and flexibility.

The system aims to provide users with a seamless experience for managing their financial accounts, executing transactions, applying for loans, and receiving notifications about various account activities.

Objectives:

- **Account Management**: To manage multiple account types (e.g., savings, current, loan accounts) with functionalities like deposits, withdrawals, balance inquiries, and account creation.
- **Transaction Handling**: To enable and manage financial transactions, including deposits and withdrawals, ensuring validation and security at every step.
- **Customer and Employee Management**: To handle customer information (e.g., name, contact details) and employee information (e.g., position, salary), with interfaces for both types.
- **Loan Management**: To facilitate the management of loans, including loan creation, interest calculation, and repayment tracking.
- **Notification System**: To notify users about transaction statuses, loan repayments, and other account activities, keeping users informed.
- **Adherence to SOLID Principles**: The system is designed following the SOLID principles to make the codebase modular, flexible, and easy to extend with new features.
- **Security and Data Integrity**: Ensuring secure and error-free financial transactions through proper validation, encapsulation, and abstraction.

System Description:

The **Financial Management System** is divided into several core components, each responsible for a distinct aspect of the system. The following outlines the structure and key classes of the project:

1.Classes :

The classes for the system, which define various entities such as customers, employees, accounts, transactions, loans, and notifications. Each class follows the **Single Responsibility Principle (SRP)**, focusing solely on its designated functionality.

- **Person.java**: An interface for shared attributes and methods between customers and employees, such as getName(), getEmail(), and getContactNumber().
- **Customer.java**: Represents a customer in the financial system. It holds attributes like customerId, name, and a list of associated accounts. The Customer class interacts with different account types and handles customer-related operations.
- **Employee.java**: Represents an employee of the financial institution. Attributes include employeeId, position, salary, and contactDetails. Employees manage and interact with customers' accounts.
- **Account.java**: A base class for various account types, including SavingsAccount, CurrentAccount, and others. It manages attributes such as accountNumber, balance, and accountHolder.
- **SavingsAccount.java**: A subclass of Account representing a savings account. It includes specific functionality for interest calculation and withdrawal restrictions.
- **CurrentAccount.java**: A subclass of Account representing a current account, with overdraft functionality and no withdrawal restrictions.
- **Loan.java**: Represents a loan, including attributes such as loanId, loanAmount, interestRate, and duration. This class also contains methods for loan repayment calculation and interest calculation.
- **SMSNotification.java**: Implements the Notification interface, responsible for sending transaction or account-related notifications to customers via SMS.
- **Deposit.java**: Represents a deposit transaction. This class ensures that the deposit is properly handled and updates the balance of the relevant account.
- **Withdraw.java**: Represents a withdrawal transaction. This class checks that the withdrawal amount does not exceed the available balance or violate specific account rules (e.g., withdrawal restrictions for savings accounts).
- **BankActivity.java**: A utility class that logs and records all activities and events in the system, such as deposits, withdrawals, and account updates.

2. Main Class:

- **Main.java**: This is the entry point of the application. It manages the flow of operations by interacting with various components of the system. Users can initiate actions like

creating accounts, making deposits, applying for loans, and checking balances from this central class.

Design Principles:

The **Financial Management System** follows the **SOLID** principles to maintain high-quality code that is easy to extend, modify, and test.

- **Single Responsibility Principle (SRP)**: Each class is designed to have a single responsibility. For instance, the Loan class is responsible only for loan calculations, while the Deposit and Withdraw classes are responsible for handling specific transaction types.
- **Open/Closed Principle (OCP)**: The system is designed to be open for extension but closed for modification. For example, new account types (e.g., BusinessAccount) or transaction types (e.g., TransferTransaction) can be added without modifying the existing code.
- **Liskov Substitution Principle (LSP)**: The system ensures that subclasses like SavingsAccount and CurrentAccount can be used interchangeably without breaking the system. All derived classes inherit common behaviors from the base Account class.
- **Interface Segregation Principle (ISP)**: The system uses small, specific interfaces (e.g., Notification) that clients (e.g., SMSNotification) implement only the necessary methods. This keeps the codebase clean and modular.
- **Dependency Inversion Principle (DIP)**: High-level modules, such as Transaction and LoanManager, depend on abstractions (Notification, Transaction) rather than concrete classes, making the system flexible and adaptable to changes.

Conclusion:

The **Financial Management System (FMS)** provides a robust solution for managing financial operations, including account management, transactions, loan management, and notifications. By adhering to **SOLID principles**, the system ensures high-quality, maintainable, and scalable code that is easy to extend and adapt as new requirements arise.

The modular architecture, which separates concerns into different classes and packages, allows for easy maintenance and future updates. The use of JUnit tests ensures that the system functions correctly and reliably under different scenarios.

In conclusion, the **FMS** serves as a secure, efficient, and adaptable solution for managing financial operations, with the flexibility to expand and evolve as the needs of the business or users grow. The adherence to **best practices** in software design ensures that the system is built for long-term maintainability and user satisfaction.