Banking Management System in Java – Planned Project Documentation

# 1. Introduction

This Java-based banking management system simulates real-world banking operations including management of savings and current accounts, fixed deposits (FDs), loans with features like balloon repayment, insurance services, transaction handling, and admin-level control features. It is designed with both educational and real-world applicability in mind, illustrating practical applications of Java programming, object-oriented principles, and database integration.

# 2. System Modules

## User Account Management

Manages user registration, authentication, and profile updates. Supports user verification and password encryption.

## Savings & Current Accounts

Supports opening, managing, and tracking balances in savings or current accounts with interest application and withdrawal limits.

## Fixed Deposits (FD)

Users can create FDs with specified amounts, tenure, and interest rates. Once an FD is created, the deposit amount cannot be altered. To modify it, users must prematurely close the FD (which may incur penalties) or open a new one. This upholds the fixed-term nature of such instruments.

## Loan Management

Allows users to apply for loans, calculate EMIs, and support flexible repayment plans including balloon repayment schemes.

## Insurance Services

Provides functionality to buy and manage insurance policies with premium calculation based on user risk factors.

## Transaction System

Processes deposits, withdrawals, and fund transfers. Logs transactions for audit and tracking.

## Authentication

Implements secure login, password hashing, and user session management.

## Admin Controls

Grants admin access to monitor user activity, approve loan requests, and manage system data.

## Report Generation

Generates detailed reports on user accounts, transactions, loans, and insurance for internal use.

## Data Persistence (PostgreSQL via JDBC)

All system data is stored securely using PostgreSQL through JDBC for reliable and scalable data access.

# 3. Technologies Used

• Java (OOP-based design)  
  
• JDBC (Java Database Connectivity):  
 JDBC is an API that allows Java applications to connect to relational databases like PostgreSQL.  
 It provides access via interfaces like Connection, Statement, PreparedStatement, and ResultSet.  
 JDBC enables executing SQL queries, retrieving results, and managing database sessions.  
 Example URL format: jdbc:postgresql://localhost:5432/bank\_system  
  
• PostgreSQL for storing all data

# 4. Computer Science Concepts Used

This banking system uses key software engineering concepts for real-world scalability and maintainability.

🔹 OOP Concepts:

• Encapsulation: Uses private fields with public getters/setters  
• Inheritance: Account/Insurance class hierarchies  
• Polymorphism: Method overloading/overriding for calculation methods  
• Abstraction: Abstract interfaces for banking services

🔹 Data Structures:

• ArrayList: Store user accounts and transactions  
• HashMap: Map user IDs to objects  
• Queue (optional): Handle request/approval flows  
• LinkedList (optional): Store transaction logs dynamically

🔹 Design Patterns:

• Singleton: Manage a single DB connection instance  
• Factory: Create objects for accounts, insurance types  
• DAO (Data Access Object): Separate data logic from business logic  
• Strategy (optional): Plug in different EMI calculators

# 5. Calculation Logic Used

• EMI (Equated Monthly Installment):

EMI = [P × R × (1 + R)^N] / [(1 + R)^N – 1]

Where:  
 P = Loan Principal, R = Monthly Interest Rate, N = Tenure in Months

• FD Maturity:

Maturity = P × (1 + r/12)^n

Where:  
 P = Principal, r = Annual Interest Rate, n = Tenure in Months

• Insurance Premium:

Premium = (Sum Assured × Risk Factor × Duration) / 100

Risk factors include:  
 • Age, Gender, Health Profile, Pre-existing Conditions  
 • Lifestyle (e.g., smoker), Location, Occupation  
 • Policy term and sum assured  
 These are evaluated using actuarial tables approved by IRDAI.

• Balloon Loan (Flexible Repayment Plan):

A non-uniform payment method with partial repayments and a large final payment.  
 Example:  
 ₹10,000 loan @ 10% monthly interest  
 Regular plan: ₹1,100/month × 10 months  
 Balloon plan:  
 Month 1: ₹2,500  
 Month 2: ₹3,500  
 Month 3: Pays 90% of balance  
 Month 4: Final 10%  
 Interest is recalculated monthly based on unpaid principal.

# 6. Conclusion

This Java banking system demonstrates real-world financial modeling using object-oriented programming, data structures, design patterns, and integration with a relational database. It can be further extended to enterprise-level applications using RESTful APIs, cloud storage solutions, and mobile platforms for broader accessibility and feature-rich banking services.