A logo with a design on it

Description automatically generated

**National University Of Modern Languages Islamabad**

**Group Members:**

**Mueez Ahmad**

**Faizan Ahmer**

**Muhammad Zubair Khan**

**Muhammad Ali Hassan Khan**

**Date: 21/05/2025**

**BankIslami ATM Simulator - Detailed Implementation Documentation**

**1. Overview of Implementation**

The **BankIslami ATM Simulator** was developed to simulate basic ATM functionalities, including:

* **Authentication**: Card number and MPIN validation.
* **Transactions**: Deposits, withdrawals, fast cash transactions.
* **Account Management**: MPIN change, balance inquiry, and mini statement retrieval.
* **Database Connectivity**: Interaction with a MySQL database for secure transaction storage.

Each module was designed using **Java Swing for the UI** and **JDBC for database operations**. Code implementation ensures input validation and transaction integrity while providing a seamless ATM-like experience.

**2. Code Structure Overview**

The project consists of several core classes:

* **Login.java** - Handles user authentication.
* **Signupone.java, Signuptwo.java, Signupthree.java** - Handles account creation.
* **Transactions.java** - Central hub for selecting transaction types.
* **Deposit.java, Withdrawal.java, Fastcash.java** - Handles financial transactions.
* **Balanceenquiry.java** - Retrieves account balance.
* **Ministatement.java** - Displays the last ten transactions.
* **Pinchange.java** - Handles MPIN change functionality.
* **Connections.java** - Establishes and maintains database connectivity.

Each class interacts with **bank\_transactions**, **Login**, and **SignupThree** tables in the MySQL database to execute transactions and manage user data.

**3. Code Comments for Each Module**

Below is a summary of how code comments were added:

**Login.java**

* **Legal Comments**: Ensuring card number and MPIN follow security standards.
* **Informative Comments**: Clarifying button actions and database queries.
* **TODO Comments**: Suggested refactoring for modularizing database calls.

**Signup Process (Signupone, Signuptwo, Signupthree)**

* **Legal Comments**: Validating CNIC format and mandatory fields.
* **Informative Comments**: Explaining logic for dropdown lists and random form number generation.
* **TODO Comments**: Suggesting enhancements in field validation.

**Transactions.java**

* **Legal Comments**: Ensuring withdrawals do not exceed balance.
* **Informative Comments**: Detailing actions associated with each button.
* **TODO Comments**: Potential improvements for UI responsiveness.

**Deposit.java & Withdrawal.java**

* **Legal Comments**: Enforcing deposit and withdrawal limits.
* **Informative Comments**: Clarifying database query logic.
* **TODO Comments**: Optimizing database transactions for better efficiency.

**Balanceenquiry.java**

* **Legal Comments**: Handling concurrency during balance retrieval.
* **Informative Comments**: Displaying balance with formatting.
* **TODO Comments**: Suggesting caching mechanism for frequent queries.

**Ministatement.java**

* **Legal Comments**: Masking sensitive user information.
* **Informative Comments**: Clarifying how transactions are sorted.
* **TODO Comments**: Improving UI to prevent overlapping issues.

**Pinchange.java**

* **Legal Comments**: Securing MPIN update mechanism.
* **Informative Comments**: Ensuring database consistency.
* **TODO Comments**: Addressing MPIN retrieval issues (see Refactoring section).

**4. Code Refactoring and Identified Code Smells**

Refactoring was conducted to improve readability, efficiency, and maintainability. The following **code smells** were identified:

**1. Bloaters - Large Methods**

* **Issue**: Some methods contained excessive code, making them difficult to maintain.
* **Solution**: Methods were split into smaller helper functions to improve readability.

**2. Inheritance Issues**

* **Issue**: UI logic was duplicated across multiple transaction-related classes.
* **Solution**: Considered creating a **BaseTransaction** class to streamline common functionality.

**3. Change Preventers**

* **Issue**: Hardcoded values for transaction limits and input validation.
* **Solution**: Introduced **configurable constants** to allow easier modifications.

**4. Dispensable Code**

* **Issue**: Redundant database connections opened multiple times.
* **Solution**: Ensured **single database connection reuse** for efficiency.

**5. Couplers - Excessive Dependencies Between Classes**

* **Issue**: Some transaction processing classes directly manipulated UI elements.
* **Solution**: Encapsulated UI logic separately, ensuring better separation of concerns.

**Fix for MPIN Change Issue**

* **Problem**: MPIN was **not fetching correctly**, causing the update to fail.
* **Reason**: The **old MPIN was being retrieved incorrectly** due to improper SQL query execution.
* **Refactoring Solution**:
  + Modified **MPIN retrieval query** to correctly fetch the associated MPIN before performing an update.
  + Implemented **validation checks** to prevent mismatches and inconsistencies.

**5. Summary of Improvements**

**Improved readability**: Refactored large methods into smaller functions.  
**Enhanced security**: Validations strengthened to prevent unauthorized transactions.  
**Optimized performance**: Reduced redundant database queries.  
**Fixed MPIN Change Bug**: Corrected MPIN retrieval logic.  
**Better modularity**: Reduced dependency between UI and backend processing.