High Level Design (HLD)

ATM Interface in Java (Console Based Application)

Date: 27-12-2024

Final Date of Revision: 29-12-2024

Document Version Control

Date Issued	Description	Author
27-12-2024	Created Initial HLD	Sohel Datta
28-12-2024	Revised HLD with Process Flow Chart	Sohel Datta
29-12-2024	Final Revision	Sohel Datta

Contents

Document Version Control 2

Abstract 4

1.Introduction 5

1.1 Why this High-Level Design Document? 5

1.2 Scope 5

1.3 Definitions 5

2 General Description 6

2.1 Project Perspective 6

2.2 Problem Statement 6

2.3 Proposed Solution 6

2.4 Further Improvements 7

2.5 Technical Requirements 7

2.6 Data Requirements 7

2.7 Libraries Used 7

3 Design Details 8

3.1 Process Flow Chart 8

3.1.1 Model Training & Evaluation 9

3.1.2 Deployment Process 9

4 Performance Analysis 9

4.1 Reusability 9

4.2 Application Compatibility 9

4.3 Resource Utilization 9

5 KPI (Key Performance Indicators) 10

6 Conclusion 10

7 References 10

Abstract

The **ATM Interface in Java** is a console-based application that simulates the core functionalities of an Automated Teller Machine (ATM). Designed for educational purposes, this project demonstrates fundamental object-oriented programming (OOP) concepts in Java, such as encapsulation, inheritance, and abstraction. The application is user-friendly, interactive, and mimics real-world banking operations in a simplified manner. This project highlights the practical implementation of Java constructs like Scanner for user input, ArrayList for transaction tracking, and modular code for better scalability. Additionally, it underscores the importance of secure user authentication and error handling in real-world applications. The **ATM Interface** serves as an ideal beginner-friendly project, providing insights into Java programming, basic financial operations, and interactive user experience design within a console-based application framework.

1. Introduction

1.1. Why this High-Level Design Document?

The purpose of this HLD is to provide a comprehensive overview of the ATM Interface project, detailing its architecture, components, and functionalities at a high level. This document is essential for aligning stakeholders' expectations and guiding developers during the implementation phase.

1.2. Scope

The scope of this project includes the development of a console-based ATM interface that allows users to perform basic banking operations securely and efficiently. The application will support user authentication, transaction management, and real-time feedback on user actions.

1.3. Definitions

Term	Definitions
ATM	**Automated Teller Machine; a device that allows users to perform banking transactions without direct human assistance.**
User ID	**A unique identifier for each account holder.**
User PIN	**A personal identification number used to authenticate the user.**
Account Holder	**The individual who owns a bank account and uses the ATM services.**
Bank Transaction	**Any operation performed by the user such as withdrawal, deposit, or transfer of funds.**

2. General Description

The ATM Interface is designed to simulate a typical ATM experience within a console application environment. Users will interact with the system through text-based menus and prompts, allowing for intuitive navigation of available banking operations.

2.1 Project Perspective

The project consists of five primary classes:

1. Account Holder: Represents the user of the ATM who holds an account.
2. Account: Manages account details such as balance and transaction history.
3. Bank Transaction: Handles various banking operations like deposits and withdrawals.
4. Bank: Represents the bank that manages multiple accounts.
5. ATM: The interface through which users interact with their accounts.

2.2 Problem Statement

There are five different classes in the project, namely, account holder, account, bank transaction bank, and particular ATM of the bank. start running the program, you will be prompted with user id and user pin. If you entered it successfully, then you unlock all the functionalities which typically exist in an ATM. The operations you can perform in this project are show transactions history, withdraw, deposit, transfer and quit.

2.3 Proposed Solution

To address the problem statement, the ATM Interface will implement a secure authentication process followed by a menu-driven interface that allows users to perform various banking operations seamlessly.

2.4 Further Improvements

Future enhancements may include:

- Integration with a database for persistent data storage.
- Enhanced security features such as two-factor authentication.
- Implementation of multi-user support.

2.5 Technical Requirements

- **Programming Language:** Java (version 8 or higher)
- **Development Environment:** Any Java IDE (e.g., IntelliJ IDEA, Eclipse)
- **Libraries:** Standard Java libraries; no external libraries required.
- **Operating System:** Cross-platform compatibility (Windows, macOS, Linux)
- **Hardware Requirements:** Any device capable of running Java applications.

2.6 Data Requirements

- User account information (user ID, PIN)
- Transaction history (date, time, amount)
- Current balance information

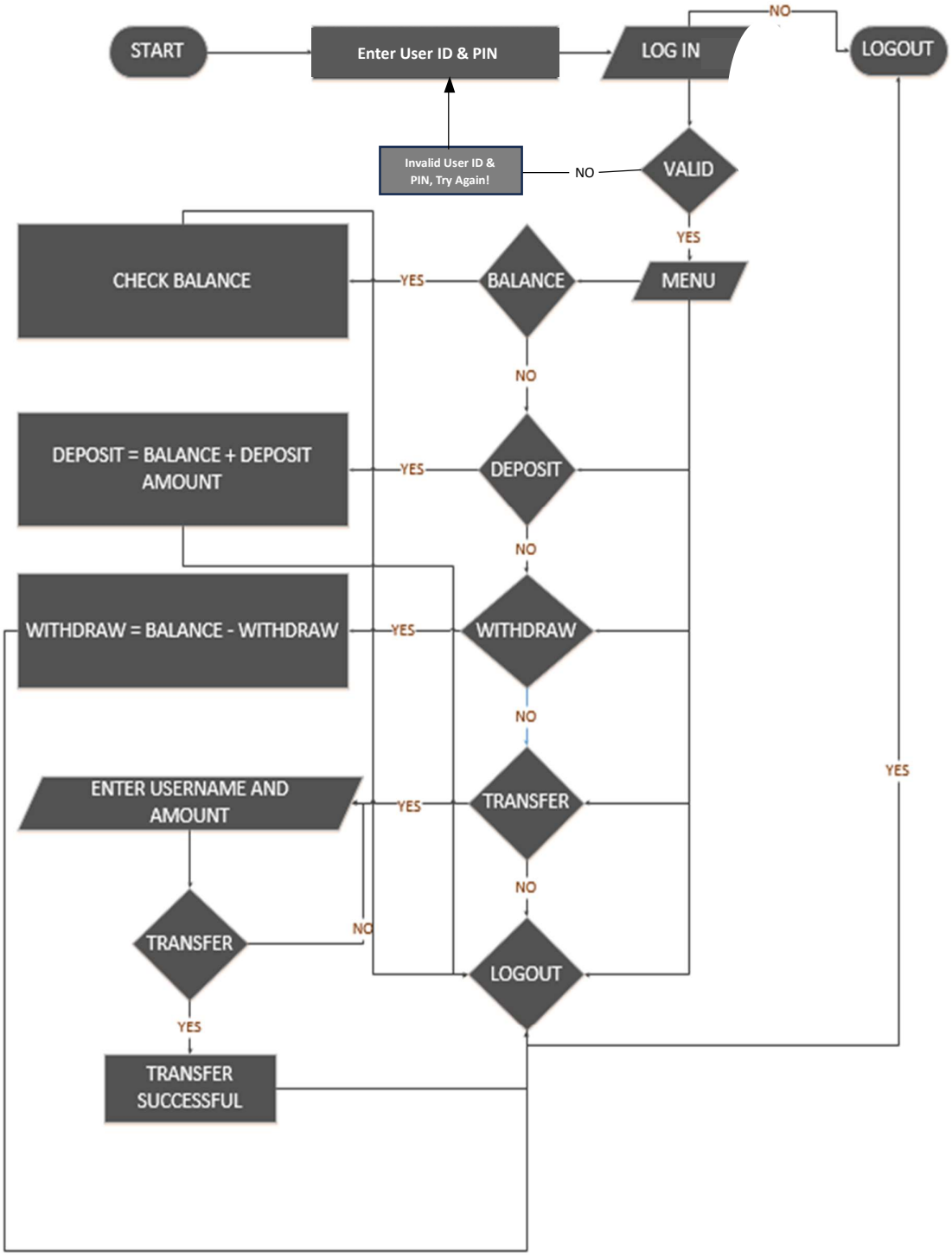
2.7 Libraries Used

The ATM Interface project primarily utilizes standard Java libraries to implement its functionalities. Below is a list of the libraries and their purposes:

- **Java Standard Library:**
  - java.util.Scanner: This library is used for obtaining input from the user through the console. It allows for reading various types of input, such as integers and strings, which are essential for user interactions like entering user IDs, PINs, and transaction amounts.
- **Java Collections Framework (if applicable):**
  - java.util.ArrayList: If the project includes features for maintaining a list of transactions or account holders, this library can be used to dynamically manage collections of objects.
  - java.util.HashMap: This can be utilized for mapping user IDs to account data, allowing for efficient lookups during authentication and transaction processing.
- **Java I/O (if applicable):**
  - java.io.File: If the project includes functionality for saving transaction history or account details to a file, this library would be used for file handling operations.

3. Design Details

3.1 Process Flow Chart



3.1.1 Model Training and Evaluation

It focuses on creating a robust application structure. The evaluation of the application will be based on:

- Unit Testing: Each module will undergo unit testıng to ensure individual components function correctly.
- Integration Testing: The interaction between modules will be tested to ensure seamless operation across the application.

3.1.2 Deployment Process

- Environment Setup: Ensure that Java Development Kit (JDK) is installed on the target machine.
- Build the Application: Compile the Java files into bytecode using the Java compiler.
- Run the Application: Execute the main class from the command line or an IDE to start the ATM interface.
- User Documentation: Provide documentation for users on how to operate the application effectively

4. Performance Analysis

4.1 Reusability

The modular design allows for easy reuse of code components in future projects or enhancements. Classes such as Account and Transaction can be utilized in other banking applications without significant modifications.

4.2 Application Compatibility

The application is developed using standard Java libraries, ensuring compatibility across different operating systems (Windows, macOS, Linux). This allows it to run seamlessly in various environments.

4.3 Resource Utilization

The application is lightweight and efficient, consuming minimal system resources due to its console-based nature. It does not require extensive memory or processing power, making it suitable for deployment on low-spec devices.

5. KPI (Key Performance Indicators)

- **User Satisfaction Rate**: Measure user feedback through surveys to assess satisfaction with the application's usability and performance.
- **Transaction Success Rate**: Track the percentage of successful transactions (withdrawals, deposits) versus failed attempts due to errors or insufficient funds.
- **Average Response Time**: Monitor the average time taken for each operation (e.g., withdrawal, deposit) to ensure quick response times.
- **Error Rate**: Calculate the number of errors encountered during transactions relative to total transactions to identify areas needing improvement.

6. Conclusion

The ATM Interface in Java Console-Based Application serves as a fundamental project that simulates essential banking operations through a console interface. By following structured methodologies and focusing on performance metrics, this project aims to provide a reliable and user-friendly experience for managing banking transactions.

Future enhancements could include integrating a graphical user interface (GUI) and expanding functionalities based on user feedback.

7. References

- CopyAssignment - https://copyassignment.com/atm-management-system-project-in-java/
- GeeksforGeeks - https://www.geeksforgeeks.org/java-program-to-display-the-atm-transaction/
- Javatpoint - https://www.javatpoint.com/atm-program-java
- Studocu - https://www.studocu.com/in/document/institute-of-engineering-science-ips-academy/computer-science/atm-interface-in-javaconsole-based-application-java/48226729