High Level Design (HLD)
Banking Application in Java (Console Based Application)

Document Version Control

Date Issued	Version	Description	Author
22/02/2023	1	Initial HLD - V1.0	Aniket Narode

Contents

Abstract

A banking application in Java can be implemented as a console-based application. The purpose of the application is to provide basic banking functionalities such as creating a new account, depositing, withdrawing, and checking account balance. Here's an abstract example of how the application can be structured:

1. Deposit Method: The deposit method will take an amount as input and add it to the account balance and print it to the console

2. Withdraw Method: The withdraw method will take an amount as input and deduct it from the account balance. The method will also check if the account has enough balance to withdraw the amount.

3. Send Money Method: The send money method will take two account objects and an amount as input. The method will first check if the sender account has enough balance to transfer the amount. If yes, it will deduct the amount from the sender account and add it to the receiver account.

1 Introduction

1.1 Why this High-Level Design Document?

The purpose of this High-Level Design (HLD) Document is to add the necessary detail to the current project description to represent a suitable model for coding. This document is also intended to help detect contradictions prior to coding, and can be used as a reference manual for how the modules interact at a high level.

The HLD will:

- Present all of the design aspects and define them in detail
- Describe the user interface being implemented
- Describe the hardware and software interfaces
- Describe the performance requirements
- Include design features and the architecture of the project
- List and describe the non-functional attributes like:
    - Security
    - Reliability
    - Maintainability
    - Portability
    - Reusability
    - Application compatibility
    - Resource utilization
    - Serviceability

1.2 Scope

The HLD documentation presents the structure of the system, such as the database architecture, application architecture (layers), application flow (Navigation), and technology architecture. The HLD uses non-technical to mildly-technical terms which should be understandable to the administrators of the system.

2 General Description

2.1 Product Perspective

a banking application is a software product that enables users to manage their finances, access banking services, and perform various financial transactions online.

2.2 Problem statement

Develop a console-based banking application that allows users to perform basic banking transactions such as depositing money, withdrawing money, transferring funds, and checking their account balance.

2.3 PROPOSED SOLUTION

To develop a console-based banking application, we can use a programming language like Java, which provides features to create a user-friendly interface and handle data securely.

2.4 FURTHER IMPROVEMENTS

1. Choose a frontend framework: There are many frontend frameworks available such as React, Angular, and Vue.js that can be used to create modern and responsive user interfaces. Choose a framework that is best suited for your development team's skillset and the application's requirements.
2. Develop a new frontend: Develop a new frontend that interacts with the existing backend API of the banking application. This new frontend should be designed with a responsive layout that is optimized for desktop and mobile devices. It should also provide a modern look and feel, with intuitive navigation and easy-to-use controls.
3. Implement new features: Take advantage of the new frontend framework to add new features to the application. For example, you could implement real-time updates for account balances or use a chatbot to help users with common banking tasks.
4. Testing: Perform rigorous testing to ensure that the new frontend works as expected and is compatible with the existing backend API. Conduct functional testing, regression testing, and user acceptance testing to ensure that the new frontend meets the requirements of the application.

2.5 Technical Requirements

1. Programming Language: Choose a programming language that is suitable for developing a console-based application. For example, Java or Python can be used to develop a console-based banking application.
2. Database Management System: Use a database management system (DBMS) to store user data, transaction history, and other relevant information. For example, MySQL or MongoDB can be used as a DBMS for the banking application.
3. User Interface: Design a user interface that allows users to perform basic banking transactions such as depositing money, withdrawing money, transferring funds, and checking their account balance. Use a console library in the chosen programming language to create a user interface that is easy to use and navigate.
4. Security: Implement security measures to protect user data from unauthorized access or theft. Use encryption techniques to secure sensitive information such as passwords and account numbers.
5. User Authentication: Implement a user authentication system that verifies the identity of the user before allowing them to access their account information.
6. Input Validation: Validate user input to ensure that data entered is in the correct format and within the acceptable range. For example, if a user tries to withdraw more money than they have in their account, the application should display an error message and prevent the transaction from occurring.
7. Transaction Management: Implement a transaction management system that allows users to deposit and withdraw money from their account, transfer funds to other accounts, and view their account balance. The application should also provide users with a transaction history, which displays all their previous transactions.
8. Error Handling: The application should be designed to handle errors gracefully. For example, if a user enters an incorrect password, the application should display an error message and prompt the user to enter the correct password.
9. Testing: Perform rigorous testing to ensure that the application works as expected and meets the requirements of the banking application.

.

2.6 Data Requirements

1. User Information: The application needs to store information about each user, such as their name, address, phone number, email address, and account number.
2. Account Information: The application needs to store information about each user's bank account, such as the account type, account balance, interest rate, and transaction history.
3. Transaction Information: The application needs to store information about each transaction that takes place, such as the transaction type (deposit, withdrawal, transfer), amount, date, and time.
4. Security Information: The application needs to store sensitive information such as user passwords, which should be encrypted to protect them from unauthorized access.
5. ATM Information: If the application includes an ATM locator feature, it will need to store information about each ATM location, such as the location name, address, phone number, and operating hours.
6. Payment Information: If the application includes a bill payment feature, it will need to store information about each user's bills, such as the billing company name, account number, and due date.

2.7 Tools used

1. Eclipse is an Integrated Development Environment (IDE) that provides a comprehensive environment for writing, testing, and debugging Java code, making it a popular choice for developing Java applications, including console-based banking applications.
2. Java is a popular programming language for developing banking applications due to its robustness, platform independence, and security features. Java can be used to develop console-based applications using libraries such as Java's built-in java.io and java.util libraries
3. The JVM is a key component in the Java platform, which enables Java applications to run on different operating systems and hardware platforms without any modifications. This makes Java a popular choice for developing cross-platform applications, including console-based banking applications.
4. Overall, Eclipse, Java, and the JVM are powerful tools for developing console-based banking applications that are robust, scalable, and secure.

2.8 Constraints

1. User Interface Limitations: A console-based application has a limited user interface, which may not be suitable for complex operations or tasks that require graphical representation.
2. Limited Functionality: A console-based application may have limited functionality compared to a full-fledged banking application, which may not meet the needs of all users.
3. Security: Console-based applications may be more vulnerable to security threats compared to web or mobile applications. It is important to implement robust security measures to protect user data.
4. Compatibility: Console-based applications may not be compatible with all operating

systems or hardware configurations, which may limit the number of users who can access the application.

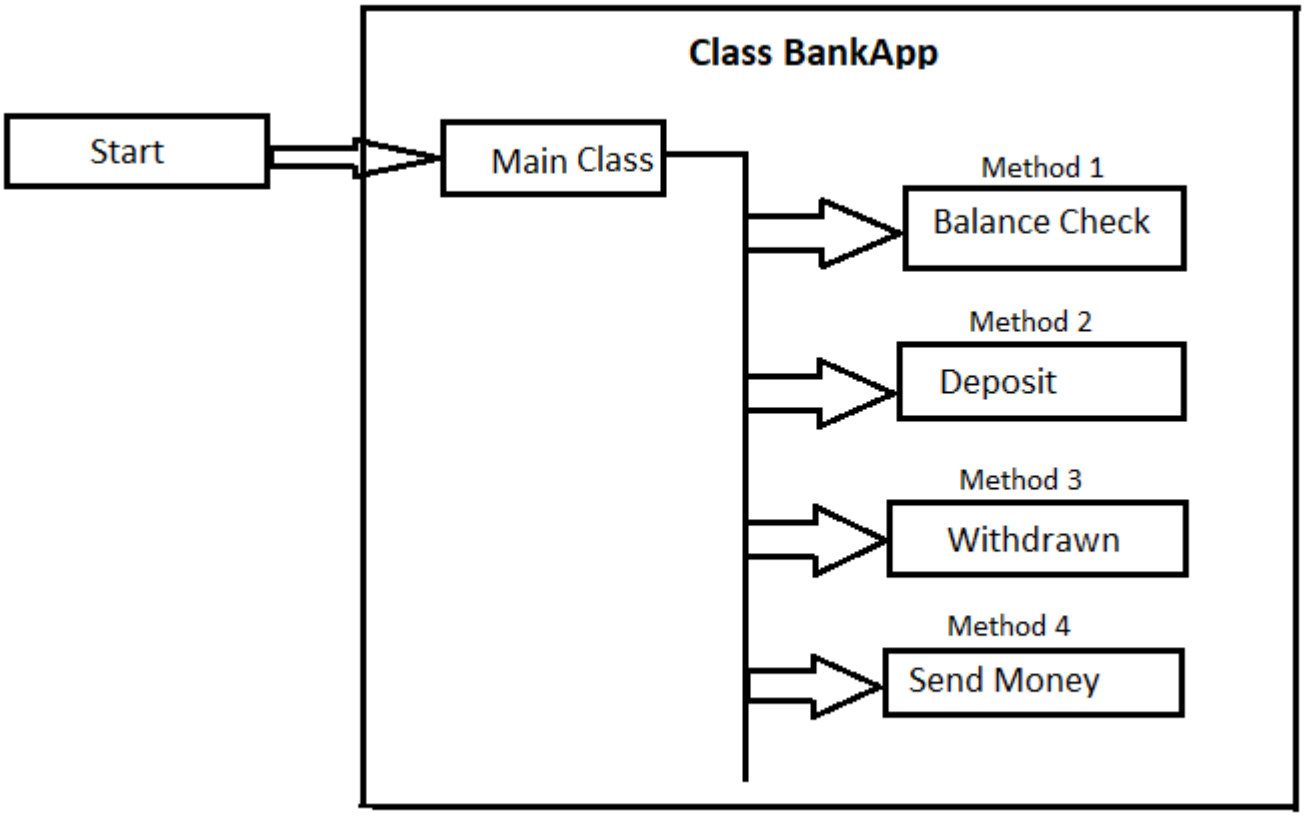
5. Usability: Console-based applications require users to have basic knowledge of commands and syntax, which may be difficult for some users to understand or learn.
6. Scalability: Console-based applications may not be easily scalable, which may limit the number of users or transactions that can be processed at once.
7. Maintenance: Console-based applications may require more maintenance compared to web or mobile applications, as they rely on the underlying operating system and hardware configuration.

2.9 Assumptions

1. Basic User Knowledge: It is assumed that users of the console-based banking application have basic knowledge of commands and syntax to interact with the application.
2. Limited Functionality: The application assumes that users are only interested in basic banking functions such as checking balances, transferring funds, and paying bills.
3. Single User Access: The application assumes that only one user can access the application at a time.
4. No Network Connection: It is assumed that the console-based banking application will run on a local machine without a network connection to ensure security.
5. No Graphical User Interface: The application assumes that users do not require a graphical user interface for basic banking functions.
6. Standard Operating System: The application assumes that the underlying operating system is standard and compatible with the tools and libraries used to develop the application.

3 Design Details

3.1 Process Flow



3.2 Event log

The system should log every event so that the user will know what process is running internally.

Initial Step-By-Step Description:

1. User Authentication: The user starts by entering their login credentials, which include their username and password. The application verifies the user's credentials and grants access to the account if the login is successful.
2. Account Overview: Once the user is authenticated, the application displays an overview of their account, including the account balance, transaction history, and other relevant information.
3. Deposit: The user can choose to deposit funds into their account by entering the amount they wish to deposit. The application prompts the user to enter the deposit amount and updates the account balance accordingly.
4. Withdrawn : The user can choose to withdraw funds from their account by entering the amount they wish to withdraw. The application checks if the account has enough balance to make the withdrawal and updates the account balance accordingly.

5. Send Money: The user can choose to transfer funds to another account by providing the recipient's account number and the amount they wish to transfer. The application checks if the sender's account has enough balance to make the transfer and deducts the amount from the sender's account while adding it to the recipient's account.

3.3 Error Handling

Should errors be encountered, an explanation will be displayed as to what went wrong? An error will be defined as anything that falls outside the normal and intended usage.

4 Performance

Performance is an important consideration for any software application, including a Java console-based banking application. Here are some factors that can impact the performance of such an application:

1. Memory Usage: The amount of memory used by the application can impact its performance. Java is a memory-managed language, so it is important to ensure that the application is not using too much memory, which can slow it down and even cause it to crash.

2. Processing Power: The processing power of the computer running the application can also impact performance. For example, if the computer is low on processing power, the application may take longer to perform certain tasks.

3. Database Performance: A banking application typically requires a database to store account information and transaction data. The performance of the database can impact the overall performance of the application. It is important to ensure that the database is optimized for performance and can handle the expected number of transactions.

4. Network Latency: If the application requires network connectivity, such as for accessing online banking services or for sending/receiving funds, network latency can impact performance. It is important to ensure that the application is designed to minimize network latency and can handle slow or intermittent connections.

5. Code Efficiency: The efficiency of the code used in the application can also impact its performance. It is important to ensure that the code is well-written and optimized for performance, using techniques such as caching and minimizing I/O operations.

4.1 Reusability

The code written and the components used should have the ability to be reused with no problems.

4.2 Application Compatibility

The different methods for this project will be using java as an interface between them. Each method will have its own task to perform, and it is the job of the java to ensure proper transfer of information.

4.3 Resource Utilization

When any task is performed, it will likely use all the processing power available until that function is finished.

5 Conclusion

In conclusion, a console-based banking application is a simple yet effective way to provide basic banking functions to users who prefer a command-line interface. Developing such an application requires careful consideration of the technical requirements, data requirements, and constraints.

Tools such as Eclipse, Java, and the JVM can be used to develop console-based banking applications that are robust, scalable, and secure. However, the limitations of the console-based user interface, compatibility issues, and security threats must be carefully considered during development.

6 References

1. https://www.javatpoint.com/
2. https://www.eclipse.org/downloads/.