CLI-Based Banking System: Variant 3

Group 8: Loop Legends

Avnoor Ludhar, Makhsuma Khamzaliyeva, Aleksa Vucak

University of Windsor

COMP2120: Obj-Oriented Programming Using Java

Dr. Olena Syrotkina

03/08/2024

# Member Contributions

|  |  |  |
| --- | --- | --- |
| Members | Files Implemented | Contributions |
| Avnoor Ludhar | -Account.java  -CheckingAccount.java  -SavingsAccount.java | **Abstract Class and Inheritance**: Implemented the Account abstract class, defining common properties and methods for all account types. This class encapsulates shared attributes such as accountNumber, accountHolder, balance, and a list of transactions.  **CheckingAccount Class**: Extended the Account class to implement the CheckingAccount class. This class overrides abstract methods like deposit and withdraw to provide specific functionality for checking accounts.  **SavingsAccount Class**: Extended the Account class to implement the SavingsAccount class, adding methods for interest calculations.  **Method Implementations**: Implemented the abstract methods such as deposit and withdraw, ensuring that they are appropriately overridden in derived classes.  **Admin Class**: Implemented the admin class for admin operations such as report generation and monitoring accounts.  **Comments and Documentation**: Added detailed comments to enhance code readability and maintainability. |
| Makhsuma Khamzaliyeva | -User.java  -Transactions.java  -Bank.java | **User and Transaction Classes**: Implemented the User class to represent bank customers, encapsulating user details like full name, username, address, and phone number. Implemented the Transaction class to represent financial transactions.  **Transaction Handling**: Developed the performTransaction method in the Bank class, enabling users to deposit and withdraw funds. Implemented exception handling to manage errors such as invalid inputs and insufficient funds.  **Account Management**: Implemented the openAccount method, allowing users to open various types of bank accounts. Designed the infrastructure and data structures (using HashMap and ArrayList) to store and manage account information.  **GenerateAccountNumber Method:** Implemented the generateAccountNumber method to generate a unique account number.  **Comments and Documentation**: Added detailed comments to enhance code readability and maintainability. |
| Aleksa Vucak | -BankSystem.java  -Bank.java  -User.java | **Main Method**: Developed the main method to manage user interactions, allowing users to perform banking operations via the command-line interface.  **Report Generation**: Implemented the generateReport method in the Bank class, utilizing file I/O to write account summaries and transaction details to a text file.  **User Registration and Transfers**: Developed the registerCustomer and registerNewCustomer methods to facilitate customer registration. Implemented the transfer method for fund transfers between accounts.  **Menu Display**: Created the showMainMenu method to display options and guide users through available features.  **Bank Class Secure Type Checking:** Implemented generics in the Bank class for secure type checking.  **Comments and Documentation**: Added detailed comments to enhance code readability and maintainability. |

Additional Member Contributions

|  |  |
| --- | --- |
| Members | Additional Contributions |
| Avnoor Ludhar | **Admin Sign In**: To utilize the admin actions from the main menu the user must enter their admin password. Upon proper entry of the password, it unlocks admin specific commands.  **ConsolePrinter Class:** Implemented ConsolePrinter class to synchronize console printing from different threads. |
| Makhsuma Khamzaliyeva | **Search Functionality**: Developed the searchAccounts method to search for accounts by full name, username, or account number.  **Exception Handling:** Handled all exceptions and implemented custom exceptions, such as the InsufficientFundsException for more robust exception handling. |
| Aleksa Vucak | **Interest Calculation**: Implemented the addInterest method in the SavingsAccount class, utilizing multi-threading to apply interest to savings accounts periodically.  **Updating Account Information**:Developed a menu choice for users to update account information. Changes to be made to the users account information include their address, and phone number. |

CLI-Based Banking System: Variant 3

The CLI-Based Banking System is a Java application that simulates basic banking operations such as opening accounts, performing transactions, and generating reports. The system supports different types of bank accounts, including checking and savings accounts, and features multi-threading for handling interest calculations. Exception handling ensures the robustness of the application, and input/output operations are used to generate reports and display account details.

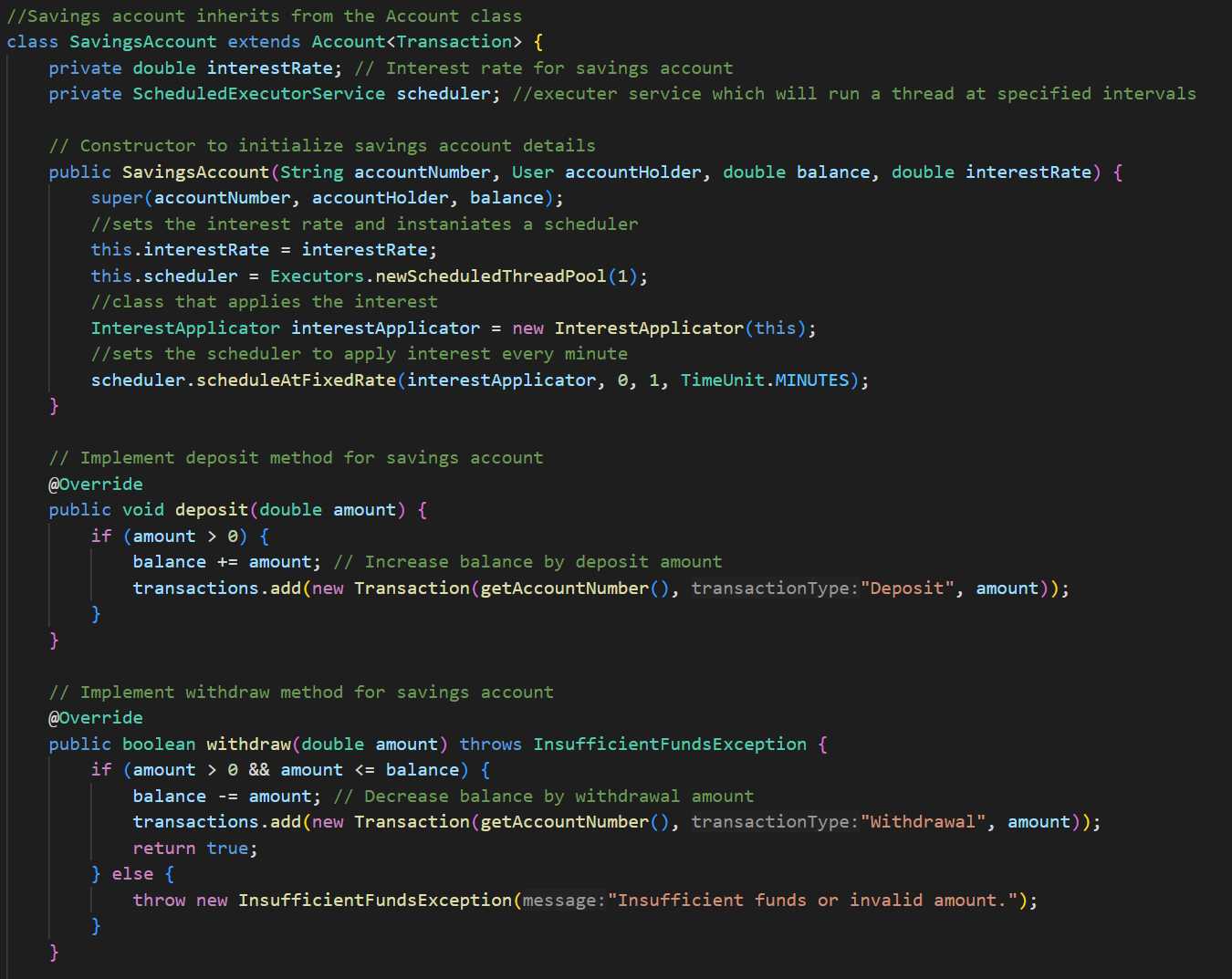
Along with our members’ contributions as listed in the chart above, we also implemented additional features that helped to enhance our program for better efficiency and utility. The interest rate application in ‘SavingsAccount’ is a feature where the savings account automatically applies interest using a dedicated thread. This feature simulates the periodic application of interest, enhancing realism in the banking simulation. Next, with the enhanced search functionality, users can search for accounts using multiple criteria, improving user experience and accessibility. Furthermore, asynchronous report generation was implemented through the use of threads while also designing a CLI menu for user interaction. Finally, transaction error handling is a comprehensive exception handling for invalid transaction types, non-existent accounts, and insufficient funds, ensuring a robust and user-friendly system.

## Inheritance and Polymorphism

Inheritance and polymorphism are crucial object-oriented programming concepts used extensively in our bank management system. Inheritance is represented by the Bank class which uses the generic type A, allowing it to handle different types of accounts (e.g., CheckingAccount, SavingsAccount). The Account class, from which CheckingAccount and SavingsAccount inherit, defines common properties and methods for all accounts. This setup leverages inheritance to ensure code reuse and maintainability. Polymorphism is demonstrated in the performTransaction method, where the method dynamically handles various types of transactions (Deposit, Withdrawal, Transfer) based on the account type. The method calls account-specific implementations (e.g., withdraw, deposit methods) through the Account interface, showcasing polymorphism.



***Figure 1.1:*** *Example showcasing inheritance and polymorphism*



***Figure 1.2:*** *Example showcasing inheritance and polymorphism*

A screen shot of a computer code

Description automatically generated

***Figure 1.3:*** *Example showcasing inheritance and polymorphism*

### Generics

Generics are used to create classes, interfaces, and methods with placeholders for types, enabling type safety and code reusability. In our system, generics are used in the Bank class definition as shown in the figure below.

A screenshot of a computer screen

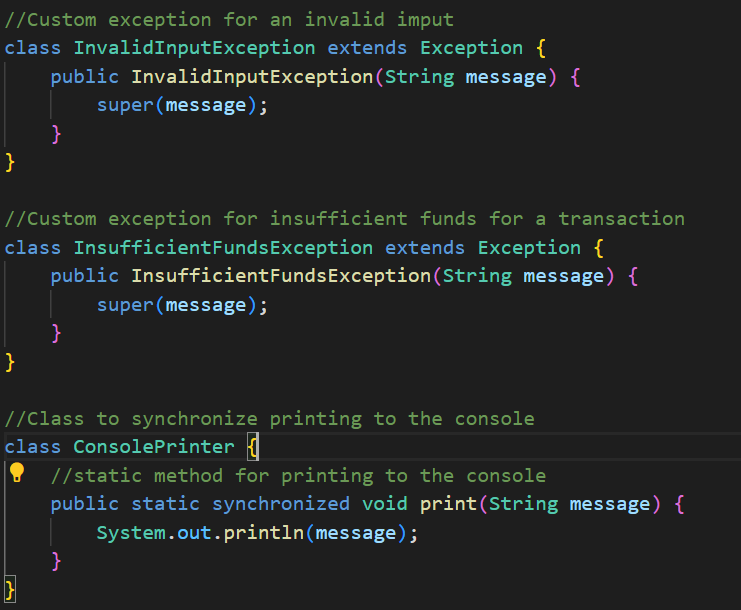
Description automatically generated

***Figure 2:*** *Example showcasing generics*

This generic type A allows the Bank class to handle any type of Account object, making it flexible for different account types (e.g., CheckingAccount, SavingsAccount) while maintaining type safety.

### Exception Handling

Exception handling is used to manage and respond to runtime errors in a controlled manner. Our system employs custom exceptions (InsufficientFundsException, InvalidInputException) to handle specific error conditions such as insufficient funds and invalid user input. This approach provides clear error messages and maintains robustness.



***Figure 3.1:*** *Example showcasing exception handling*

*A black background with text

Description automatically generated*

***Figure 3.2:*** *Example of two exceptions being thrown*

### Collections and Data Structures

Our system utilizes various collections and data structures to manage data. The Map interface (specifically HashMap) is used to store and retrieve accounts and users efficiently. The Map is ideal for key-value pair storage, providing fast lookups by account number or username.

A screenshot of a computer program

Description automatically generated

***Figure 4:*** *Example showcasing the use of a ‘Map’*

**Multithreading**

Multithreading is used to perform operations concurrently, improving responsiveness and performance. In our system, the InterestApplicator periodically applies interest to all savings accounts in the bank. This approach ensures that interest calculations and updates are handled concurrently with other operations.

A screen shot of a computer program

Description automatically generated

***Figure 5:*** *Example showcasing the use of multithreading in*

### I/O Programming

I/O programming is employed to handle file operations in our system. The generateReport method demonstrates file writing using FileWriter to create a report. This method ensures that the report is saved asynchronously and handles exceptions related to file operations.

A screen shot of a computer program

Description automatically generated

***Figure 6:*** *Example showcasing report generation with ‘FileWriter’*

### Conclusion

This report has detailed the implementation of key object-oriented and concurrent programming concepts in the provided Java banking system. Through well-structured examples and thorough explanations, we have showcased the utilization of inheritance and polymorphism to extend and customize account types, the application of generics for type safety in managing collections, and the importance of robust exception handling for maintaining system integrity. Furthermore, the report highlighted the strategic use of collections to dynamically handle data, demonstrated multithreading to efficiently manage periodic tasks like interest application, and illustrated I/O programming for persistent data storage. Each of these elements contributes to creating a comprehensive, efficient, and scalable banking system. By understanding and leveraging these features, developers can further enhance the functionality and performance of software applications, ensuring they are well-equipped to handle complex, real-world requirements.

References

Syrotkina, O. (2024). Introduction to Java Programming [PowerPoint slides]. School of Computer Science, University of Windsor. https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/986367/View

Syrotkina, O. (2024). Objects and Classes in Java [PowerPoint slides]. School of Computer Science, University of Windsor. <https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/989954/View>

Syrotkina, O. (2024). Inheritance, Polymorphism, and Generics [PowerPoint slides]. School of Computer Science, University of Windsor. <https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/994505/View>

Syrotkina, O. (2024). Exception Handling [PowerPoint slides]. School of Computer Science, University of Windsor. https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/999558/View

Syrotkina, O. (2024). Collections and Data Structures [PowerPoint slides]. School of Computer Science, University of Windsor. <https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/1001793/View>

Syrotkina, O. (2024). Multithreading [PowerPoint slides]. School of Computer Science, University of Windsor. <https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/1016705/View>

Syrotkina, O. (2024). I/O Programming [PowerPoint slides]. School of Computer Science, University of Windsor. https://brightspace.uwindsor.ca/d2l/le/content/170433/viewContent/1016703/View