



ATM Simulation

Vu Hoang Phuc 20214923

ACKNOWLEDGEMENT

I would like to express our appreciation to Dr. Tran Van Dang who gave me the opportunity to work on this project. This work could not be finished without your helpful advices and thorough guidance.

Contents

1	Introduction	3
2	Components	4
2.1	Account Class	4
2.2	Bank Class	4
2.3	User Interaction	4
3	Implementation	6
3.1	Dictionary for Account Management	6
3.2	Deque for Transaction History	6
3.3	Priority Queue (Heap) for Daily Reset Scheduling	7
3.4	Timestamp-Based Cooldown Enforcement	7
3.5	JSON Serialization for Persistence	7
4	Result	8
4.1	Account Management	8
4.2	Check Balance	8
4.3	Deposit Funds	10
4.4	Withdraw Funds	10
4.5	Transaction History	10
4.6	Quit	11
5	Conclusion	13

Chapter 1

Introduction

The ATM Simulator is designed to replicate the core functionalities of an automated teller machine, providing a practical platform for users to manage their banking transactions securely and efficiently. With the increasing reliance on digital banking solutions, this simulation serves to educate users about the principles of account management, transaction limits, and the importance of financial discipline. By implementing real-world banking scenarios, the simulator allows users to navigate functions such as deposits, withdrawals, and transaction history while adhering to predefined constraints. This hands-on experience encourages informed decision-making regarding personal finance.

The primary motivation behind developing the ATM Simulator is to create a robust educational tool that demystifies banking operations and enhances financial literacy. It leverages modern programming techniques and data structures—such as dictionaries for account management, deques for transaction history, and heaps for scheduling daily resets—to ensure optimal performance and user experience. By engaging users in a simulated environment, the ATM Simulator not only provides a safe space for practicing financial transactions but also fosters an understanding of the underlying algorithms and data structures that support banking systems. This initiative aims to empower users with the knowledge necessary to navigate the complexities of personal finance in today's digital economy.

Chapter 2

Components

The ATM Simulator is structured around three primary components: the `Account`, the `Bank`, and the user interface. Each component plays a crucial role in simulating real-world banking operations, allowing users to manage their finances effectively.

2.1 Account Class

The `Account` class is fundamental to the simulator, encapsulating the attributes and behaviors associated with a bank account. Each instance of the `Account` class represents a user's account and includes properties such as account ID, balance, and transaction limits for withdrawals and deposits. The class also tracks daily transaction counts and totals, ensuring compliance with set limits. Methods within the class, such as `withdraw`, `deposit`, and `get_transaction_history`, enable users to perform transactions while maintaining a history of their activities. The use of a deque for transaction history ensures that users can efficiently access their most recent transactions.

2.2 Bank Class

The `Bank` class serves as the central management system for handling multiple accounts. It maintains a dictionary of accounts, allowing for quick retrieval and management of user data. The class includes methods for creating new accounts, loading and saving account data to JSON files, and processing daily resets of transaction limits using a min-heap. This heap efficiently schedules daily reset events, ensuring that transaction limits are refreshed at the right time. Additionally, the `Bank` class enforces cooldown periods between transactions, preventing users from executing rapid-fire operations and thus mimicking the realistic constraints of banking systems.

2.3 User Interaction

The user interaction is facilitated through a console-based menu system in the `main` function. Users are prompted to enter their account ID (acting as a PIN), which the `Bank` class uses to retrieve the corresponding `Account` instance. The menu provides options for checking balance, withdrawing, depositing, viewing transaction history, or quitting the application. This interactive approach allows users to simulate

real banking operations while the underlying code manages the interactions between the user, the Bank, and the Account objects seamlessly.

Through this structured design, the ATM Simulator effectively combines object-oriented programming principles with practical banking functionalities, providing users with an engaging platform to learn and practice financial management.

Chapter 3

Implementation

The ATM Simulator is designed to handle various banking operations efficiently. Below is a deeper exploration of the data structures and algorithms integrated into the simulator, highlighting their functionalities and advantages.

3.1 Dictionary for Account Management

The `Bank` class utilizes a dictionary (`dict`) to manage user accounts effectively.

- **Efficiency:** With an average time complexity of $O(1)$ for operations such as lookup, insertion, and deletion, the dictionary allows for quick access to account information. This is crucial for high-frequency banking operations, where every millisecond counts.
- **Scalability:** The dictionary can grow dynamically, accommodating thousands of accounts without significant performance degradation. This scalability is essential as the user base expands.
- **Implementation:** Each account is represented as an `Account` object, which contains relevant attributes such as balance, transaction history, and limits. This structured approach makes it easier to manage and access account data.

3.2 Deque for Transaction History

The use of a double-ended queue (`collections.deque`) for transaction history management brings several benefits:

- **Constant-Time Operations:** The deque allows for $O(1)$ time complexity for both appending new transactions and popping old ones, making it ideal for maintaining a fixed-size history.
- **Memory Efficiency:** By limiting the size of the deque, the application can effectively manage memory usage, storing only the most recent transactions. This is particularly important for users who perform many transactions.
- **Functionality:** The deque supports operations like transaction reversal and retrieval of recent activity, providing users with a clear audit trail. This functionality enhances user experience and trust in the banking system.

3.3 Priority Queue (Heap) for Daily Reset Scheduling

To manage transaction limits and cooldowns, a min-heap (`heapq`) is implemented:

- **Optimized Scheduling:** The priority queue efficiently tracks the next reset times for each account, ensuring that operations like insertion and extraction occur in $O(\log N)$ time. This optimization is crucial for maintaining performance as the number of accounts grows.
- **Focused Operations:** By only processing accounts nearing their reset times, the system minimizes unnecessary computations. This targeted approach reduces overhead and enhances overall performance.
- **Real-Time Compliance:** The priority queue ensures that transaction limits are reset as required, helping maintain compliance with banking regulations and user expectations.

3.4 Timestamp-Based Cooldown Enforcement

The ATM Simulator employs timestamps to regulate the cooldown periods between transactions:

- **Lightweight Mechanism:** By storing the timestamp of the last completed transaction within each `Account` object, the system can quickly assess whether a user is eligible for another transaction. This avoids the need for complex state management.
- **Realistic Banking Delays:** The cooldown enforcement mimics real-world banking processes, where users cannot execute transactions in rapid succession. This feature adds a layer of realism to the simulator.
- **Implementation:** The use of `datetime` for timestamp management allows for precise calculations and comparisons, ensuring that cooldown periods are enforced accurately.

3.5 JSON Serialization for Persistence

The application leverages JSON serialization for data persistence:

- **Hierarchical Data Representation:** While JSON is not a traditional data structure, it effectively represents hierarchical data, making it suitable for storing complex account information and transaction histories.
 - **Data Continuity:** By saving account states in JSON format, the simulator can restore previous sessions seamlessly. This feature is vital for users who expect their banking data to remain consistent over time.
 - **Ease of Integration:** JSON's widespread use and compatibility with various programming languages and frameworks allow easy integration into the application, simplifying data storage and retrieval processes.
-

Chapter 4

Result

4.1 Account Management

Account management is the backbone of the ATM Simulator, allowing users to create and maintain individual accounts. Each account is identified by a unique ID (typically a PIN) and comes with personalized settings such as daily transaction limits, maximum amounts per transaction, and allowable transaction frequencies. These configurations ensure that users can practice financial discipline while working within real-world constraints. Account details, including balance and transaction history, are stored securely, providing a realistic and personal banking experience. Additionally, the system dynamically schedules daily resets for transaction limits, ensuring seamless operation without manual intervention.

4.2 Check Balance

The balance inquiry feature offers users an instant view of their current financial status. By selecting this option, users can check their account balance, which is updated dynamically after every deposit or withdrawal. This functionality mimics the convenience of real-world ATMs, where users can quickly verify their funds before making decisions. It empowers users to stay informed about their finances, ensuring they remain within their spending limits and maintain control over their accounts. This simple yet essential feature underpins the importance of regular financial monitoring.

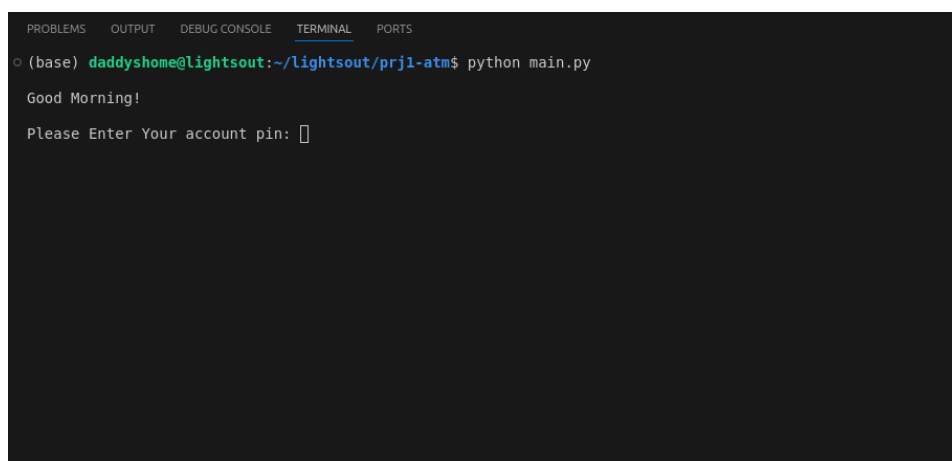
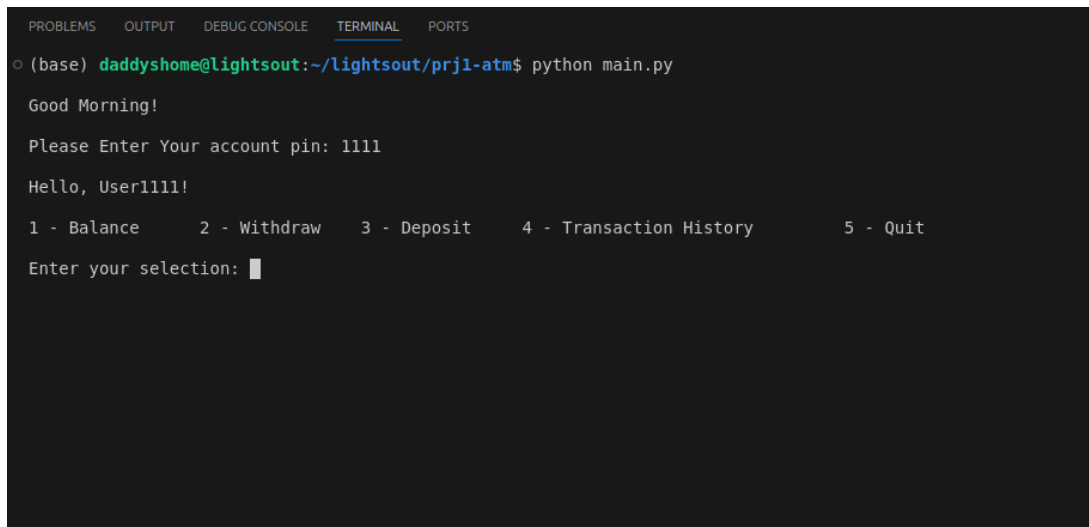
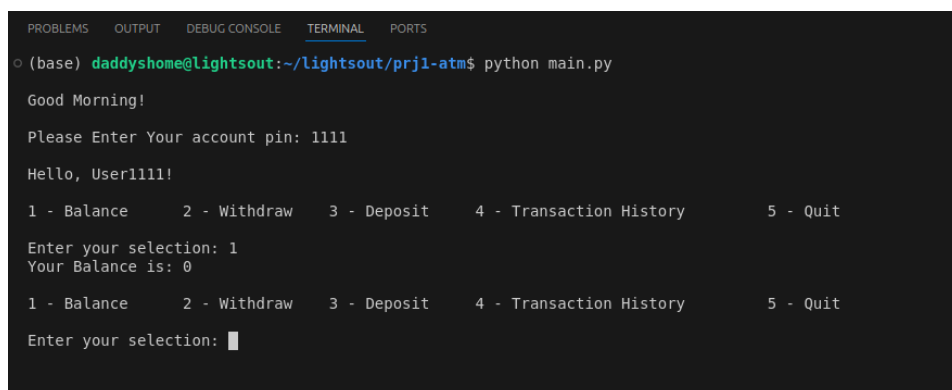
A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tabs, the terminal shows the command prompt '(base) daddyshome@lightsout:~/lightsout/prj1-atm\$' followed by the command 'python main.py'. The output of the command is 'Good Morning!' followed by a prompt 'Please Enter Your account pin: ' with a small square cursor.

Figure 4.1: Welcome Menu

A terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The terminal shows the execution of a Python script. The prompt is (base) daddyshome@lightsout:~/lightsout/prj1-atm\$. The user runs python main.py. The program outputs "Good Morning!", "Please Enter Your account pin: 1111", and "Hello, User1111!". It then displays a menu: "1 - Balance", "2 - Withdraw", "3 - Deposit", "4 - Transaction History", and "5 - Quit". The prompt "Enter your selection:" is shown with a cursor.

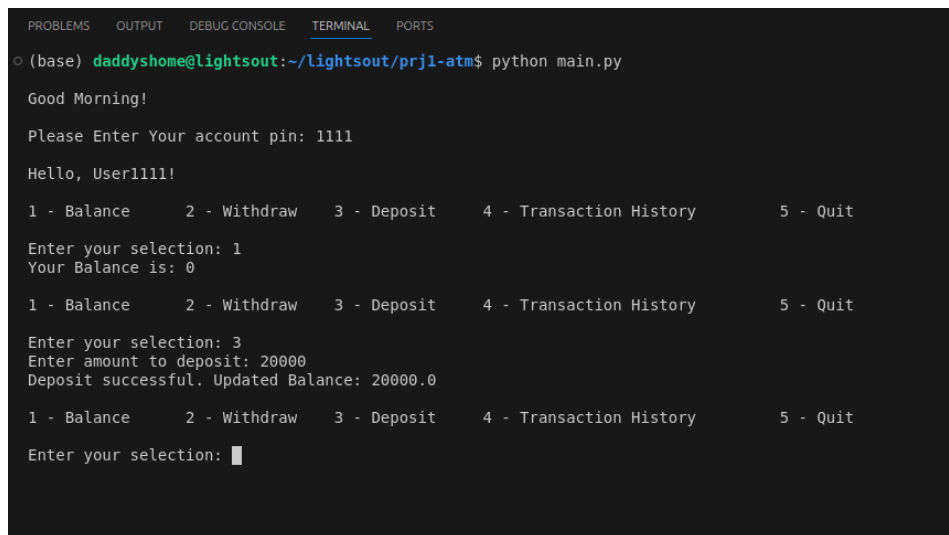
```
(base) daddyshome@lightsout:~/lightsout/prj1-atm$ python main.py
Good Morning!
Please Enter Your account pin: 1111
Hello, User1111!
1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History      5 - Quit
Enter your selection: █
```

Figure 4.2: Logging into an Account

A terminal window showing the continuation of the program. The user enters '1' for the selection. The program outputs "Your Balance is: 0". The menu is displayed again, and the prompt "Enter your selection:" is shown with a cursor.

```
(base) daddyshome@lightsout:~/lightsout/prj1-atm$ python main.py
Good Morning!
Please Enter Your account pin: 1111
Hello, User1111!
1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History      5 - Quit
Enter your selection: 1
Your Balance is: 0
1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History      5 - Quit
Enter your selection: █
```

Figure 4.3: Checking Balance



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(base) daddyshome@lightsout:~/lightsout/prj1-atm$ python main.py
Good Morning!
Please Enter Your account pin: 1111
Hello, User1111!
1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 1
Your Balance is: 0
1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 3
Enter amount to deposit: 20000
Deposit successful. Updated Balance: 20000.0
1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: █
```

Figure 4.4: Depositing Funds

4.3 Deposit Funds

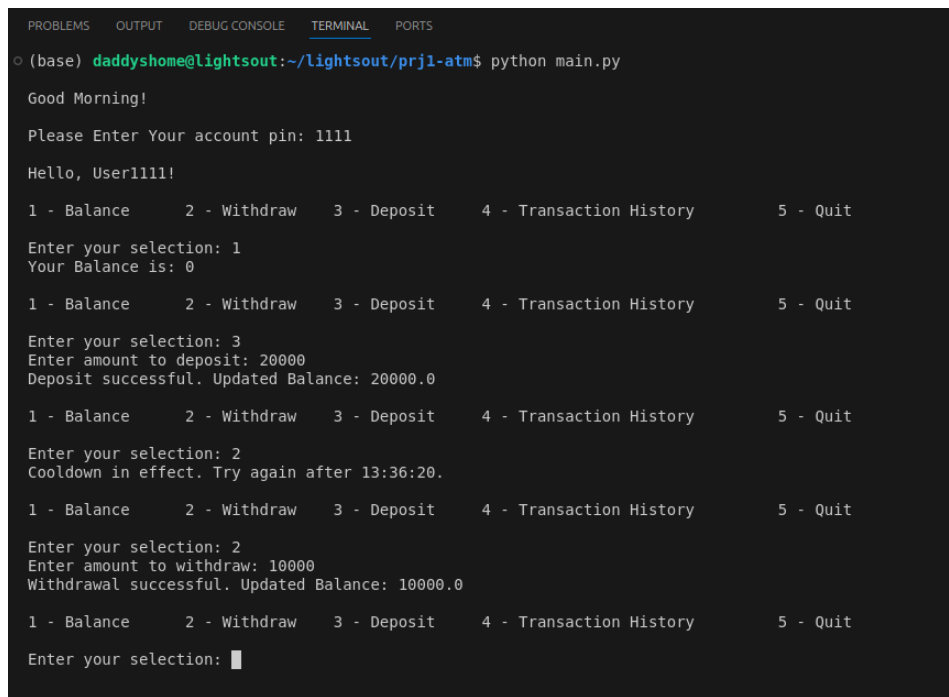
Depositing funds is equally crucial for account management, enabling users to add money to their accounts. The simulator enforces per-transaction and daily deposit limits, as well as a maximum number of deposits allowed per day. This teaches users to manage their deposits thoughtfully while ensuring they do not exceed predefined caps. When a deposit is successful, the account balance updates, and the transaction is logged for future reference. Failed deposits provide immediate feedback on the violated constraint, enhancing the learning experience. By simulating these real-world conditions, the deposit functionality promotes financial responsibility.

4.4 Withdraw Funds

The withdrawal functionality lets users access their funds while adhering to realistic banking constraints. Users can withdraw money within predefined per-transaction and daily limits, ensuring responsible financial behavior. Each withdrawal also considers the user's current balance, preventing overdrafts. The system tracks the number of withdrawals per day and disallows further transactions if the frequency cap is exceeded. If a withdrawal fails due to exceeding these constraints, users receive clear feedback, helping them understand the importance of staying within limits. Successful withdrawals are logged in the transaction history, providing a transparent record for reference.

4.5 Transaction History

The transaction history feature allows users to view a log of their recent financial activities, offering a clear picture of their account usage. Each record includes the transaction type (deposit or withdrawal), the amount involved, the updated account balance, and a timestamp for reference. The system uses a double-ended queue (deque) to efficiently store up to 10 transactions, automatically removing the oldest ones as new records are added. This concise yet informative history enables users to track their spending



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
○ (base) daddyshome@lightsout:~/lightsout/prj1-atm$ python main.py

Good Morning!

Please Enter Your account pin: 1111

Hello, User1111!

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 1
Your Balance is: 0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 3
Enter amount to deposit: 20000
Deposit successful. Updated Balance: 20000.0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 2
Cooldown in effect. Try again after 13:36:20.

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 2
Enter amount to withdraw: 10000
Withdrawal successful. Updated Balance: 10000.0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: █
```

Figure 4.5: Withdrawing Funds

and deposits over time, fostering better financial planning and accountability.

4.6 Quit

The simulator ensures that users' progress is preserved by saving all account data before exiting. When users select the quit option, their account details, including balances, transaction histories, and settings, are serialized into a JSON file. This allows the simulator to load the accounts seamlessly during the next session, ensuring continuity. The ability to save and restore accounts highlights the importance of data persistence in real-world financial systems. By providing this functionality, the simulator gives users confidence that their practice sessions and financial decisions remain intact across multiple interactions.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Please Enter Your account pin: 1111
Hello, User1111!

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 1
Your Balance is: 0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 3
Enter amount to deposit: 20000
Deposit successful. Updated Balance: 20000.0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 2
Cooldown in effect. Try again after 13:36:20.

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 2
Enter amount to withdraw: 10000
Withdrawal successful. Updated Balance: 10000.0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 4

Recent Transactions:
- 2025-01-19 13:36:05: deposit $20000.0 (Balance: $20000.0)
- 2025-01-19 13:36:41: withdraw $10000.0 (Balance: $10000.0)

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: █
```

Figure 4.6: Transaction History

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Hello, User1111!

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 1
Your Balance is: 0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 3
Enter amount to deposit: 20000
Deposit successful. Updated Balance: 20000.0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 2
Cooldown in effect. Try again after 13:36:20.

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 2
Enter amount to withdraw: 10000
Withdrawal successful. Updated Balance: 10000.0

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 4

Recent Transactions:
- 2025-01-19 13:36:05: deposit $20000.0 (Balance: $20000.0)
- 2025-01-19 13:36:41: withdraw $10000.0 (Balance: $10000.0)

1 - Balance      2 - Withdraw    3 - Deposit      4 - Transaction History    5 - Quit
Enter your selection: 5
Thank you for banking with us!
○ (base) daddyshome@lightsout:~/lightsout/prj1-atm$ █
```

Figure 4.7: Logging out

Chapter 5

Conclusion

In summary, the ATM Simulator serves as an innovative educational tool that effectively demonstrates the intricacies of banking operations through a user-friendly interface. By leveraging object-oriented programming principles, the simulator encapsulates the essential functionalities of a real-world ATM, allowing users to engage in transactions while adhering to realistic constraints. The design, featuring the Account and Bank classes, ensures efficient management of user data and transaction limits, while the interactive console interface enhances user experience. Ultimately, this simulator not only provides a platform for practicing financial management but also fosters a deeper understanding of the underlying algorithms and data structures that power modern banking systems, empowering users to navigate their financial journeys with confidence.