Chem274B Final Project – DeliveryII

Software Engineering Reflection

Name: Trinity Ho

Group: Group 6

Team Members: Trinity Ho, Priscilla Vaskez, Dongwan Kim

1. Role in the Project

My primary responsibility in this project was the development of methods for Levels 1, 3, and 4, as well as testing for Levels 1, 2, and 3 to ensure the correctness and robustness of the implemented system.

Level 1:
I implemented the foundational feature of the banking system with create_account() method. This method is essential for initializing new accounts. The structure of an account was determined in this step. The account initialization includes timestamp and account balances, maintaining the account dictionary structure that other methods rely on.

Level 3:
I implemented the pay() method, which handles withdrawals, records outgoing transactions data created in level 2, and assigns unique payment IDs. I also implemented the get_payment_status() method, which allows users to check payment status and whether cashback has been applied. In addition, I incorporated the helper function, _process_cashback(), which was written by a teammate, into methods to ensure that cashback rewards are correctly applied and recorded in the account balance.

Level 4:
I developed the _record_balance() helper function to track account balance changes over time whenever an operation modifies an account's balance. This function supports all methods that affect balances, ensuring historical account balance integrity. I also implemented _binary_search_record(), a helper function, for the get_balance() method, which efficiently searches the balance of an account at a specific timestamp.

2. Contribution to Project Completion

Our team divided work and collaborated across different foundational methods, helper functions, and levels. I also coordinated with the team to ensure shared data structures were used correctly and that each method aligned with the project's overall architecture

I contributed significantly to testing Levels 1, 2, and 3. Much of my testing was focused on validating the correct behavior of the _process_cashback method, ensuring that payments and cashback updates were properly reflected in account balances.

I focused on efficient algorithms for querying balance record history, implementing binary search for account balances at specific timestamps. This contributed to the system's scalability and performance for large transactions and efficiency.

3. Challenges and How We Solved Them

Implementing cashback tracking, in particular, was complex, lengthy, and difficult to read. To address this, I collaborated with my teammate and incorporated their helper function _process_cashback, which streamlined the code and reduced duplication across methods that required cashback updates.

Additionally, maintaining balance history across multiple methods was challenging, as it required careful tracking of all account changes. My teammate highlighted the complexity of implementing this across every method. To solve this, I developed the _record_balance helper function for the get_balance method and systematically integrated it into all methods that modify account balances, ensuring consistent and reliable balance tracking throughout the system.

4. Algorithmic and Performance Analysis of Each Method.

- create_account - O(1)

  This method adds a new account and starts up a balance history. This method performs constant-time insertion into the dictionary.

- deposit - O(1)

  This method looks up, updates, and records deposits at constant-time per account

- transfer - O(1)

This method updates balances of source and target accounts, records balance history, and updates outgoing totals. All operations are constant-time.

- top_spender - $O(N^2)$

  This method sorts accounts by outgoing totals using bubble sort. Time complexity grows quadratically with the number of accounts.

- pay - O(1)

  This method subtracts payment, tracks payment, and calculates cashback updates at constant-time dictionary operations.

- get_payment_status - O(1)

  This method looks up payment status in the dictionary with constant-time access.

- merge_account - O(N1 + N2)

  The method runs in linear time with respect to the total number of stored records in both accounts, because it must merge balance records, payments, and pending cashback data.

  This method
- get_balance O(log N)

  This method uses binary search to find account balance at a specific timestamp within a balance record history of an account.

5. What Could Have Been Done Differently

Software Project Management

A more consistent and detailed documentation throughout the project could have been maintained. Regular updates to the README file and clearer docstrings for each method would have helped track progress, clarify design decisions, and reduce confusion when integrating teammates' code.

Better documentation would also have made it easier to identify dependencies between methods and ensure consistent use of shared data structures, especially as the project increased in complexity across levels.

Final Product Improvements

Priority queues or heaps could replace manual sorting in methods like top_spenders, improving performance and scalability as the number of accounts grows.

More comprehensive automated testing could also be added for edge cases involving timestamps and account state transitions for further robustness.