Gerald Arocena

CSCI E-97, Fall 2019

Assignment 1

Design Document

The following is a list of some of the deviations from or variations on the design I made as well as other notes:

- I represented transactionList internally as a LinkedHashMap<transactionId, Transaction> instead of as just a "list" as specified in the API. This representation maintains insertion order and provides a quick way to find a particular transaction. I also used a LinkedHashMap for both accountBalanceMap and blockMap because knowing the order in which accounts were created could be practical and would also make for a neat representation for the blocks, especially since they have an ordering to them (i.e., blockNumber) as well. It also provides a fast way to search for a particular account or block.
- I added attributes to the Block class to store temporary states and a method called closeBlock to update them so that the Ledger can track relevant state changes as it transfers information when finishing one block and starting the next. I also added other utility methods in the Block class such as addAccount and addTransaction as a means for the ledger to respectively add/retrieve accounts and transactions to/from the Block. Actually, I ended up adding many additional utility methods throughout my implementation that I thought made intuitive sense in the overall structure of the design. I used Java doc comments to indicate where such methods are and to provide descriptions for them.
- My processCommand method runs an interactive prompt that allows the user to enter in commands one at a time.
- I added more exceptions than was specified in the requirements or sample test and my additional test file ("additional.script") was able to showcase most of these.
- I created an "id maker" attribute in the Ledger class called suggestedId (basically an incrementing counter) to reassign an id for duplicate transaction IDs passed into the ledger. Also, I used a HashSet in the ledger called transactionIdsUsed to collect all processed transaction IDs. I thought it'd be a good way to quickly find if an incoming transaction's id is a duplicate and to keep track of how many transactions were processed via HashSet's size method rather than iterating through all of the blocks.

- My getBlock implementation returns all transactions of a block regardless of whether it’s “complete” (i.e., has 10 transactions) or not. This allows us to see discrete progression of a block at various stages of transaction processing which is good for debugging, and also practical. The requirements did not specify what kind of blocks to return, e.g., incomplete or complete.
- I wasn’t sure how to trigger exception handling for the validate method in Ledger for showcasing but I used stdout in my code for debugging to check and make sure it’s working correctly. That code is still in the Ledger, inline commented as “Debugging”, but I commented it all out for submitting the assignment.
- I added the command get-total-blocks to the DSL even though it wasn’t listed as one of the commands in the API. It provides a way to get the total number of blocks in the blockchain which was a requirement stated in other parts of the design document.

The Ledger Service design was of great quality in my opinion. The design document definitely helped with my implementation probably because it was well thought out. For instance, I thought having previousHash as an attribute was sufficient enough for a block to be tied to the one before it, and that having a reference to the previous block itself was superfluous. However, I did not realize until much later on that having the previous block reference made things a lot easier, e.g., for traversing the blockMap and validating the blockchain. I also liked that the design was flexible at the same time. Considering types of objects, which utility methods to create and where, and what good abstractions are and enforcing them, among other things, all helped me to understand and appreciate the significance of the three design principles of abstraction, divide and conquer, and separation of concerns. Nonetheless, accessing accounts from Block objects sometimes felt awkward. Having a pointer directly from Ledger would also have been a viable design choice in my opinion. There were important relationships such as who is a master account and who isn’t (i.e., types of accounts) that could also have been facilitated well in such a design.