Mariam Gogia
CSCI E – 97
Ledger Service

Detailed changes / additions by each Class:

Account Class
No changes were made in Account class.

LedgerException
No changes were made in LedgerException class.

CommandProcessor
No changes were made in CommandProcessor class.

CommandProcessorException
No changes were made in CommandProcessorException class.

Transaction Class
No structural changes were made in Transaction class; however, the validity of its properties was checked in i*sValidTransaction(Transaction transaction)* method in Ledger class.
The Ledger class's i*sValidTransaction(Transaction transaction*) method is called by Ledger's *processTransaction function* and its purpose is to validate the inputs in Transaction object. *isValidTransaction(Transaction transaction)* ensures that no transaction with negative amount, fee less than 10, or note length with greater than 1024 characters gets processed.

Block Class
No changes were made in Block object's structure; however, several methods were added to the class.

boolean addTransaction (Transaction transaction)

This method adds transaction to Block class's *transactionList;*
It is called by the Ledger class in *processTransaction* function.
If *addTransaction* return false, it means that the transaction could not be added as the list is full, so it is time to commit that block.

String getHash (String seed)

This is a getter but with the signature 'seed.' With the help of its helper function *computeHash,* it computes the hash of the block as per requirement. *getHash(String seed)* is called by the Ledger's *addBlock function.*

String computeHash(String seed, String transactionToString)

This helper method computes hashes given 2 strings and it is called by *getHash(String seed)* function. It passes the seed and the nodes of the Merkle Tree and gets the hash in return.

String toString()

As requirements state, when the Block is retrieved by its number, the details of the Block should be outputted. *toString()* method converts block details into a single formatted string and is used when the *getBlock(Int blockNumber)* is called.

void addAccount (String address, Account account)

This method puts a new valid account to *accountBalanceMap*

Ledger Class

private Block currentBlock was added to Ledger's properties.
This property was needed to keep track of details of the current uncommitted block.

createAccount(String account ID)

added a functionality to throw the LedgerException if the account already exists

processTransaction(Transaction transaction)

added a helper method *isValidTransaction(Transaction transaction*) to check whether the transaction object's properties are set to valid values.
added a boolean *blockFull* checking the number of transactions in the *transactionList.* If the block is full, it calls *addBlock(currentBlock)* method to add the block to the blockchain.

Map<String, Integer> getAccountBalances()

added a functionality to throw the LedgerException if there are no committed blocks at all.

Block getBlock (int blockNumber)

added a functionality to throw the LedgerException if there is no such block.

Transaction getTransaction (String transactionID)

added a functionality to throw the LedgerException if there is no such transaction in the blockchain.

Other Notes:

1. All classes have setters and getters for all of their properties.
Most of them are never used, as per design requirements, they are not taken out from the code.

2. To my understanding, if the transaction is successfully processed, and therefore accepted, *getTransaction(int id)* function should return the transaction by its id even when the block is not committed. Which is different from other functions related to accounts and account balances. Therefore, as per my implementation, if the transaction is processed its details will be returned by *getTransaction(int id*) function

3. All my LedgerExceptions are caught and therefore the program will keep running if it ecounters LedgerExceptions; however, as I consider CommandProcessorExceptions fatal, if the program encounters unrecognizable or incorrectly formatted commands – it will throw an exception and stop.

4. The implementation is case sensitive throughout.

Comments regarding design document:

1. The design document was extremely helpful in understanding the project structure. It gave me direction and guidance which saved me some time and helped me succeed in this project.
2. The only thing that would have made the design document better is a better communicated flow of the system. We were given structure details of classes and methods but in some cases the flow - the timing of events - was unclear.