Caleb Wong
8397914
Miner ID: cwong131

CSI2110: Assignment 2

The objective of the assignment was to design a simple blockchain. Students were taught the basics of building a blockchain and learned the concepts behind cryptographic hashes and proofs of work.

We were asked to design and implement 3 java classes: Transaction, Block, and BlockChain.

In the Transaction class, the class held three attributes: sender, receiver, and amount. The goal of the Transaction class a simple class with getters, to organize attributes of transactions that would go inside each block.

In the Block class, the class holds eight attributes: index, timestamp, transaction, nonce, previousHash, hash, duration, and counter. The first six respectively were required attributes in each of our blocks, however the duration and counter were added to record the amount of time and number of trials needed to generate a hash with the appropriate amount of zeroes.

The Block class contains two constructors. One is used when reading files, the other (the one that takes a block type) is used when adding new blocks to the blockchain. In the second block constructor, the nonce and hash is generated when an appropriate hash that satisfies the conditions needed. I will describe these conditions when exploring those methods. The following methods are used for nonce generation:

findNonce(), returns a string that contains 10 random characters.
rand(), returns a random character between ASCII 33 and 127.
findHash(), returns a hash that from the toString() method that is converted by the SHA-1 algorithm. The method constantly loops until it has found a hash that begins with “00000”. In order to do this, I used a null string as my condition in the constructor. If the method finds a hash that begins with the correct number of zeroes, rather than an empty hash, the method will return a hash.

The BlockChain class contains a single constructor that takes in an ArrayList of blocks. We were asked to create 5 different methods: fromFile, getBalance, validateBlockchain, addBlock, and a toFile. In addition, a main method was created to add and validate the blockchains.

fromFile(), reads a file and goes through each line to find an index, timestamp, sender, receiver, amount, nonce, and hash, to put into a block. However if the user does not put these things in the correct order as shown in the bitCoinBank.txt file, it will incorrectly allocate the variables. After reading new variables, it will add blocks into an ArrayList of blocks and call the constructor to create a new instance of a BlockChain.

getBalance(), assumes every user has a balance of 0 goes through the blockchain looking for usernames inside each transaction. When the name appears as a sender, it will subtract, if the name appears as a receiver, it will add to their balance. getBalance() returns the new balance of the given user after going through each block.

validateBlockchain(), will validate go through an entire blockchain. When going through the first block, the first block is implemented differently because it is the genesis block and the previousHash is set to

“00000”. That is why the first block is implemented separately from the other blocks. The method will check for previousHashes, hashes, and indexes, to ensure they are correct. If any of the conditions are false, the method will return false and the blockchain would be invalid. I have added print statements to find out which line would break when verifying invalid blockchains.

addBlock(), simply adds a block to a blockchain.

toFile(), creates a file and prints out each block in the blockchain and their attributes in the same order as the example bitCoinBank.txt file.

main(), will prompt a user for a miner, and a read file. The main file will then check if the txt is a valid blockchain. If the blockchain is invalid, the program will throw an exception; but if the blockchain is valid, it will prompt for more transactions. If the user chooses to add more transactions, the program will prompt for a receiver, sender, and amount, and find a nonce that will go with the transaction. In addition, a duration and counter is given to see the amount of time and trials needed to reach the end. The user will be prompt again to make another transaction. The process repeats if the user would like to add more transactions, however if the user would not like to add additional transactions, the program writes the new blockchain into a file and the program ends.

Here are statistics on the 10 transactions.

Transaction #	Duration (nanoseconds)	# of Trials
1	2496980754	346559
2	1957213152	510107
3	5094844302	1382733
4	4149324730	1081086
5	156131807	41467
6	4904121994	1316831
7	5037571077	1361150
8	11662031204	3103125
9	1987371925	536318
10	5260143199	1418972