CST 412 – Assignment 2

In this assignment, we will implement a basic application of blockchain for a cryptocurrency as discussed in class. Please follow the instructions below to complete the activity.

Programming languages: you can choose to work with any programming language that can capture the requirements. Java is recommended for this assignment.

1. Block Class.

A blockchain is composed of several blocks. Please use the following UML diagram and description to write a class to represent a block.

Block	
- hash: String	This block's hash.
- prevHash: String	The previous block's hash.
- data: String	Data recorded in the block.
- timeStamp: long	The time when the block was created.
- pow: int	The proof of work.
- prefix: int	Number of required 0's in the block's hash with SHA-256.
+ Block(data: String, prevHash: String, timeStamp: long, prefix:int)	Constructor.
+ generateBlockHash(): String	Method to compute the block's hash.
+ mineBlock(): String	Method to mine a block.

- The constructor should initialize the values for the `data`, `prevHash`, `timeStamp`, and `prefix` with the corresponding parameter values. Then it should call the method `mineBlock()` to mine the block (get a working `pow` and update the value of `hash` with the returned value from the call). For the genesis block, the `prevHash` value should be `null`.
- **Implement all accessor and mutator methods for every field.**
- `generateBlockHash()` method: this method should first concatenate all different fields of the block to generate the hash from. Then, it should get an instance of the SHA-256 hash function and perform the hash with the concatenated string. You can use the `MessageDigest` class, under the package `java.security`, to calculate cryptographic hashing value in Java. This method returns the hash value as a string.
- `mineBlock()` method: this method should keep increasing the value of the `pow` and compute the hash value, by calling the method `generateBlockHash()`, until the hash value meets the requirement specified by the `prefix`. For example, if `prefix` = 3, there will be three leading 0's in the accepted hash value. **Please note that this number is based on the assumption that the hash values generated are in 256-bit format.** However, you can choose any format to store the hash values. Another possible format is using hex digits, and you need to make sure that it will work correctly with the prefix value under the assumption above.
  This method returns the accepted hash value as a string.

2. BlockChain Class.

After having a class for blocks, we will need to write a class to represent a blockchain using the following UML diagram and description

BlockChain	
- blocks: List<Block> - chainSize: int	The list stores all the added blocks. The current size of the blockchain.
+ BlockChain() + addBlock(newBlock:Block): void + verify(): Boolean + toString(): String	Constructor. Method to add a new block to the list. Method to verify the whole blockchain. Method to output the blockchain information in a specific format.

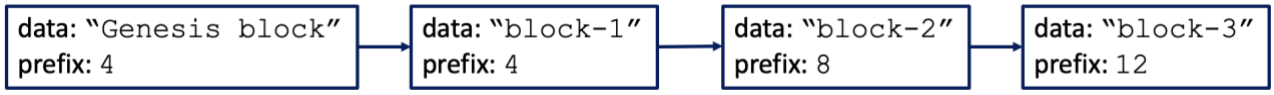
- The constructor should initialize the `blocks` with an empty `List<Block>`, and the `chainSize` with `0`.
- **Implement all accessor and mutator methods for every field.**
- `addBlock(newBlock:Block)`: this method takes a `Block` object and adds it to the end of the chain `blocks`. Please make sure to update the `chainSize` properly.
- `verify()` method: this method should go through every block in the `blocks` and check for the following:
    - The stored hash of the block is actually what it computes.
    - The hash of the previous block stored in the current block is actually the hash of the previous block, except for the genesis block.
    - The current block has been mined properly.

  This method returns `true` if all blocks in the chain are verified, and returns `false` otherwise.
- `toString()` method: this method should return a string that contains the size of the chain and information of every block in the same order as stored in the chain. For each block, include the following information: data, prefix, previous hash's block, and hash value. You should consider implementing a `toString()` method in the `Block` class and use it in this method. You should choose a readable format of the blockchain for the output string.

3. Application Class.

1. Create a new class called `TestBlockChain`, and write a `main` method to test the implemented classes. In the `main` method, create a new `BlockChain` object, and create new `Block` objects to add to the chain using the chain diagram below. Note that when creating a new block, use the current time of the machine for the `timestamp` field.

Testing chain:



2. Call the `verify` method to verify the created chain. Then print out the verification status (verified/not verified).
3. Print out the chain information.

Submission Guidelines.

Please submit the following:

1. The source code of your program.
2. A document with instructions on how to compile and run your program with command-line commands.

Compress all required files and submit them as a zip file.

Grading Rubric.

Task	**Points**
The program can compile and run successfully.	20
Clear instructions on how to run the program.	10
Well-commented codes.	10
The program produces expected outputs for the described test case in section 3 and several test cases provided by the instructor.	60
Total	**100**