Author: Shivam Patel
Andrew ID: shpatel
Email Address: shpatel@cmu.edu
Last Modified: November 1, 2022
Project 3

Project 3 – Task 0

Task 0 Execution

```
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 2000000
Expected total hashes required for the whole chain: 256.000000
Nonce for most recent block: 189
Chain hash: 004B86E27D46ABEC74476039B0B0FAC69C0D631BA03C65E50C09094F6CEA8444
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
2
Enter transaction
Alice pays Bob 100 DSCoin
Total execution time to add this block was 3 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
```

Enter difficulty > 0
2
Enter transaction
Bob pays Carol 50 DSCoin
Total execution time to add this block was 3 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
2
Enter transaction
Carol pays Donna 10 DS Coin
Total execution time to add this block was 3 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Chain verification: TRUE
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index":0,"timestamp":"2022-11-01
23:24:18.589","data":"Genesis","previousHash":"","nonce":189,"difficulty":2},{"index":1,"timestamp
":"2022-11-01 23:26:25.135","data":"Alice pays Bob 100
DSCoin","previousHash":"004B86E27D46ABEC74476039B0B0FAC69C0D631BA03C65E50C09094F6C
EA8444","nonce":116,"difficulty":2},{"index":2,"timestamp":"2022-11-01 23:27:00.458","data":"Bob
pays Carol 50
DSCoin","previousHash":"00713796D6EDF614392675F0675BC135D23132F90B705EC652BD7E1CDE8
BDC4F","nonce":152,"difficulty":2},{"index":3,"timestamp":"2022-11-01 23:27:32.596","data":"Carol
pays Donna 10 DS
Coin","previousHash":"008FBF7FB5E3A0AEF9D6FDD51E482810E0D7FB323CA19D512E48A59F8A9D

4C71","nonce":293,"difficulty":2}],"chainHash":"0059378C4DC36710E23E5936450D3798449F8418A64013902C193A4C693DEA0F"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
4
Corrupt the Blockchain
Enter block ID of block to corrupt
1
Enter new data for block 1
Alice pays Bob 76 DSCoin
Block 1 now holds Alice pays Bob 76 DSCoin
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index":0,"timestamp":"2022-11-01 23:24:18.589","data":"Genesis","previousHash":"","nonce":189,"difficulty":2},{"index":1,"timestamp":"2022-11-01 23:26:25.135","data":"Alice pays Bob 76 DSCoin","previousHash":"004B86E27D46ABEC74476039B0B0FAC69C0D631BA03C65E50C09094F6CEA8444","nonce":116,"difficulty":2},{"index":2,"timestamp":"2022-11-01 23:27:00.458","data":"Bob pays Carol 50 DSCoin","previousHash":"00713796D6EDF614392675F0675BC135D23132F90B705EC652BD7E1CDE8BDC4F","nonce":152,"difficulty":2},{"index":3,"timestamp":"2022-11-01 23:27:32.596","data":"Carol pays Donna 10 DS Coin","previousHash":"008FBF7FB5E3A0AEF9D6FDD51E482810E0D7FB323CA19D512E48A59F8A9D4C71","nonce":293,"difficulty":2}],"chainHash":"0059378C4DC36710E23E5936450D3798449F8418A64013902C193A4C693DEA0F"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Chain verification: FALSE
Improper hash on node 1. Does not begin with 00
Total execution time to verify the chain was 0 milliseconds

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
5
Total execution time required to repair the chain was 16 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Chain verification: TRUE
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
4
Enter transaction
Donna pays Sean 25 DSCoin
Total execution time to add this block was 391 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
0
Current size of chain: 5
Difficulty of most recent block: 4
Total difficulty for all blocks: 12
Approximate hashes per second on this machine: 2000000
Expected total hashes required for the whole chain: 66560.000000
Nonce for most recent block: 141061
Chain hash: 00003781BD43D52057256DDDF62F0365E49F7D7B88A30350E0661280393E7960
0. View basic blockchain status.

1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
6

Process finished with exit code 0

Task 0 Block.java

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Email: shpatel@cmu.edu
 * Last Modified: November 1, 2022
 * File: Block.java
 * Part Of: Project3Task0
 *
 * This Java file acts as a Block for the BlockChain that would be
 * created by the BlockChain class. It has a Block constructor and
 * some of its specific functions include function to calculate hashes,
 * compute proof of work, and convert the Block object to a JSON string.
 */

// Defines the package for the Java file
package org.example;

// Imports necessary for Gson, BigInteger, MessageDisgest and
NoSuchAlgorithmException
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;


public class Block {

    // Stores the index of the Block in the Blockchain
    // The first block (the so-called Genesis block) has an index of 0.
    private int index;

    // Stores the time when the Block was created
    private java.sql.Timestamp timestamp;

    // Stores the transaction on the Block
    private java.lang.String data;

    // Stores the SHA256 hash of a block's parent. This is also called a
hash pointer
    private java.lang.String previousHash;

    // Stores a BigInteger value determined by a proof of work routine
    private java.math.BigInteger nonce;

    // Stores the integer value that specifies the minimum number of left
most hex digits needed
    // by a proper hash. The hash is represented in hexadecimal.
    private int difficulty;


    // Constructor to initialise the values of the instance variables of
the Block class
    Block (int index, java.sql.Timestamp timestamp, java.lang.String data,
int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
```

```
        this.difficulty = difficulty;
        this.nonce = BigInteger.valueOf(0);
    }

    /***
     * Computes the SHA256 hash value of the following:
     * index + timestamp.toString() + data + previousHash + nonce +
difficulty
     * @return A String of the hexadecimal representation of the hash
     */
    // Source: Shivam Patel Project 1 Task 1 - CMU Heinz 95-702
    public java.lang.String calculateHash() {

        // String whose hash is to be found
        String hash_input = index + timestamp.toString() + data +
previousHash + nonce + difficulty;

        // Source: CMU 95702 Fall 2022 Lab1-InstallationAndRaft Code
        byte[] digest = new byte[0];
        try {
            // Access MessageDigest class for SHA256
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            // Compute the digest
            md.update(hash_input.getBytes());

            // Store digest as a byte array for further use
            digest = md.digest();
        }
        // Handles No SHA-256 Algorithm exceptions
        catch (NoSuchAlgorithmException e) {
            // Print error message in console
            System.out.println("No SHA-256 available" + e);
        }
        // Return the SHA256 hash in String form
        return bytesToHex(digest);
    }

    /***
     * Function to find a good hash. It increments the nonce until it
produces a good hash.
     * This method calls calculateHash() to compute a hash of the
concatenation of the index,
     * timestamp, data, previousHash, nonce, and difficulty. If the hash
has the appropriate
     * number of leading hex zeroes, it is done and returns that proper
hash. If the hash
     * does not have the appropriate number of leading hex zeroes, it
increments the nonce by
     * 1 and tries again. It continues this process, burning electricity
and CPU cycles, until
     * it gets lucky and finds a good hash.
     * @return A String with a hash that has the appropriate number of
leading hex zeroes.
     *         The difficulty value is already in the block. This is the
minimum number of hex 0's
     *         a proper hash must have.
     */
    public java.lang.String proofOfWork() {

        // Source to repeat a String multiple times:
```

```
        // https://stackoverflow.com/questions/1235179/simple-way-to-repeat-a-string
        String leading_zeros = new String(new char[difficulty]).replace("\0", "0");

        // Stores the calculated hash
        String hash;

        // While a good hash is found
        while (true)
        {
            // Calculate new hash
            hash = calculateHash();

            // If hash starts with the required number of leading zeros
            if (hash.startsWith(leading_zeros))
            {
                // End the while loop
                break;
            }
            // If hash did not start with the required number of leading zeros, increment nonce
            nonce = nonce.add(BigInteger.valueOf(1));
        }
        // Return the good hash
        return hash;
    }

    /***
     * This function overrides Java's toString method to convert the object of the Block
     * class to a JSON string
     * @return A JSON representation of all of this block's data is returned
     */
    public java.lang.String toString() {

        // Source to format date in Gson:
        // https://stackoverflow.com/questions/6873020/gson-date-format

        // Create a Gson object
        Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd HH:mm:ss.SSS").create();

        // Serialize to JSON
        return gson.toJson(this);
    }

    // Code to convert from byte array to hexadecimal String
    // Source: https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-hex-string-in-java
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();

    /***
     * Function to convert a byte array to hexadecimal String
     * @param bytes Byte array to be converted to hexadecimal String
     * @return Hexadecimal notation (in String form) of the input byte array
     */
    public static String bytesToHex(byte[] bytes) {
```

```
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
        }
        return new String(hexChars);
    }

    /***
     * Function to get transaction details of Block
     * @return Transaction details of Block
     */
    public java.lang.String getData() {
        return data;
    }

    /***
     * Function to get difficulty of Block
     * @return Difficulty of Block
     */
    public int getDifficulty() {
        return difficulty;
    }

    /***
     * Function to get index of Block
     * @return Index of Block
     */
    public int getIndex() {
        return index;
    }

    /***
     * Function to get nonce of Block
     * @return Nonce of Block
     */
    public java.math.BigInteger getNonce() {
        return nonce;
    }

    /***
     * Function to get hash of parent of Block
     * @return Hash of parent of Block
     */
    public java.lang.String getPreviousHash() {
        return previousHash;
    }

    /***
     * Function to get time of creation of Block
     * @return Time of creation of Block
     */
    public java.sql.Timestamp getTimestamp() {
        return timestamp;
    }

    /***
     * Function to set transaction details of Block
     * @param data Transaction details of Block
     */
```

```
    public void setData(java.lang.String data) {
        this.data = data;
    }

    /***
     * Function to set difficulty of Block
     * @param difficulty Difficulty of Block
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /***
     * Function to set nonce of Block
     * @param nonce Nonce of Block
     */
    public void setNonce(BigInteger nonce) {
        this.nonce = nonce;
    }

    /***
     * Function to set index of Block
     * @param index Index of Block
     */
    public void setIndex(int index) {
        this.index = index;
    }

    /***
     * Function to set hash of parent of Block
     * @param previousHash Hash of parent of Block
     */
    public void setPreviousHash(java.lang.String previousHash) {
        this.previousHash = previousHash;
    }

    /***
     * Function to set time of creation of Block
     * @param timestamp Time of creation of Block
     */
    public void setTimestamp(java.sql.Timestamp timestamp) {
        this.timestamp = timestamp;
    }
}
```

Task 0 BlockChain.java

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Email: shpatel@cmu.edu
 * Last Modified: November 1, 2022
 * File: BlockChain.java
 * Part Of: Project3Task0
 *
 * This Java file creates a BlockChain by using the objects of the
 * Block class as its blocks. It has a BlockChain constructor and
 * its specific functions include function to add a block to the
blockchain,
 * compute number of hashes per second, get Block by index, get number
 * of hashes per second, get the latest block, get time of the Block,
 * get the total difficulty of the blockchain,  get the total expected
 * hashes for the blockchain, verify the blockchain, repair the blockchain
 * if it is corrupted and finally convert the blockchain object to a JSON
 * string.
 */

// Defines the package for the Java file
package org.example;

// Imports necessary for Gson, BigInteger, MessageDisgest and
NoSuchAlgorithmException,
// TimeStamp, ArrayList, and IO operations
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

public class BlockChain {

    // Stores the blocks of the BlockChain
    final ArrayList<Block> ds_chain;

    // Stores the SHA256 hash of the most recently added Block
    String chainHash;

    // Stores the approximate number of hashes per second on this computer
    transient int hashesPerSecond;

    // Constructor to initialise the instance variables of the BlockChain
class
    // This constructor creates an empty ArrayList for Block storage. This
constructor
    // sets the chain hash to the empty string and sets hashes per second
to 0.
    BlockChain() {
        ds_chain = new ArrayList<>();
        chainHash = "";
        hashesPerSecond = 0;
    }
```

```java
    /***
     * Function to add a new Block to the BlockChain
     * @param newBlock Block to be added to the BlockChain
     */
    public void addBlock(Block newBlock) {

        // Update chainHash to be the hash of the new block that is being
added
        chainHash = newBlock.proofOfWork();

        // Add new block to the array list
        ds_chain.add(newBlock);
    }

    /***
     * This method computes exactly 2 million hashes and times how long
that process
     * takes. So, hashes per second is approximated as (2 million / number
of seconds).
     * It is run on start up and sets the instance variable
hashesPerSecond. It uses
     * a simple string - "00000000" to hash.
     */
    public void computeHashesPerSecond() {

        // Source to calculate the run time of a program:
        // https://stackoverflow.com/questions/5204051/how-to-calculate-
the-running-time-of-my-program
        long startTime = System.nanoTime();

        // Compute hashes for the string "00000000" for 2 million times
        for (int i = 1; i <= 2000000; i++) {
            computeSHA256("00000000");
        }

        long endTime = System.nanoTime();
        long totalTime = endTime - startTime;

        // Source to convert time in nanoseconds to seconds:
        // https://mkyong.com/java/java-how-to-convert-system-nanotime-to-
seconds/
        long totalTimeInSeconds = TimeUnit.SECONDS.convert(totalTime,
TimeUnit.NANOSECONDS);

        // Stores the number of hashes per second
        hashesPerSecond = (int) (2000000 / totalTimeInSeconds);
    }

    /***
     * Function to return block at position i
     * @param i Position of the Block to be returned
     * @return Block at postion i
     */
    public Block getBlock(int i) {
        return ds_chain.get(i);
    }

    /***
     * Function to get the chain hash
     * @return The chain hash.
```

```
     */
    public String getChainHash() {
        return chainHash;
    }

    /***
     * Function to get the size of the chain in blocks
     * @return The size of the chain in blocks
     */
    public int getChainSize() {
        return ds_chain.size();
    }

    /***
     * Function to get hashes per second
     * @return The instance variable approximating the number of hashes per
second
     */
    public int getHashesPerSecond() {
        return hashesPerSecond;
    }

    /***
     * Function to get a reference to the most recently added Block
     * @return A reference to the most recently added Block
     */
    public Block getLatestBlock() {
        return ds_chain.get(ds_chain.size() - 1);
    }

    /***
     * Function to get the current system time
     * @return The current system time
     */
    public java.sql.Timestamp getTime() {
        return new Timestamp(System.currentTimeMillis());
    }

    /***
     * Function to compute and return the total difficulty of all blocks
     * on the chain. Each block knows its own difficulty.
     * @return The total difficulty of the chain
     */
    public int getTotalDifficulty() {

        int totalDifficulty = 0;

        for (int i = 0; i < ds_chain.size(); i++) {
            totalDifficulty = totalDifficulty +
ds_chain.get(i).getDifficulty();
        }

        return totalDifficulty;
    }

    /***
     * Function to compute and return the expected number of hashes
required for the entire chain
     * @return The total expected hashes for the blockchain
     */
    public double getTotalExpectedHashes() {
```

```
        // Stores the total expected hashes for the chain
        double totalExpectedHashes = 0;

        // Loop over all the blocks in the blockchain and increment the
        // count of the total expected hashes
        for (int i = 0; i < ds_chain.size(); i++) {
            totalExpectedHashes = totalExpectedHashes + Math.pow(16,
ds_chain.get(i).getDifficulty());
        }

        // The total expected hashes for the chain
        return totalExpectedHashes;
    }

    /***
     * Function to check if the blockchain is valid.
     * If the chain only contains one block, the genesis block at position
0, this
     * routine computes the hash of the block and checks that the hash has
the requisite
     * number of leftmost 0's (proof of work) as specified in the
difficulty field. It
     * also checks that the chain hash is equal to this computed hash. If
either check
     * fails, return an error message. Otherwise, return the string "TRUE".
If the chain
     * has more blocks than one, begin checking from block one. Continue
checking until
     * you have validated the entire chain. The first check will involve a
computation of
     * a hash in Block 0 and a comparison with the hash pointer in Block 1.
If they match
     * and if the proof of work is correct, go and visit the next block in
the chain. At
     * the end, check that the chain hash is also correct.
     * @return "TRUE" if the chain is valid, otherwise return a string with
an appropriate error message
     */
    public java.lang.String isChainValid() {

        // Loop over all the blocks in the chain
        for (int i = 0; i < ds_chain.size(); i++) {

            // Prepare string whose hash is to be calculated
            String hash_input = ds_chain.get(i).getIndex() +
ds_chain.get(i).getTimestamp().toString()
                    + ds_chain.get(i).getData() +
ds_chain.get(i).getPreviousHash()
                    + ds_chain.get(i).getNonce() +
ds_chain.get(i).getDifficulty();

            // Compute SHA256 hash of the hash_input string prepared
            String hash = computeSHA256(hash_input);

            // Compute the number of leading zeros required in the hash
            // Source to repeat a String multiple times:
            // https://stackoverflow.com/questions/1235179/simple-way-to-
repeat-a-string
            String leading_zeros_required = new String(new
char[ds_chain.get(i).getDifficulty()]).replace("\0", "0");
```

```
            // If hash does not start with the required number of leading
zeros
            if (!hash.startsWith(leading_zeros_required)) {

                // Return error message
                return "Improper hash on node " + i + ". Does not begin
with " + leading_zeros_required;
            }

            // If chain size is greater than 1
            if (ds_chain.size() > 1) {

                // If it is not the last block
                if (i != ds_chain.size() - 1) {

                    // If hash of current block does not equal the previous
hash of the next block
                    if (!hash.equals(ds_chain.get(i +
1).getPreviousHash())) {

                        // Return error message
                        return "Hash of Block " + i + " does not match with
previous hash of Block " + (i + 1);
                    }
                }
                // It is the last block
                else
                {
                    // If hash of the last block does not equal to the
chain hash
                    if (!hash.equals(chainHash)) {

                        // Return error message
                        return "Hash of the last Block (Block " + i + ")
does not match with Chain Hash!";
                    }
                }
            }
            // If only one block is present in the chain
            else {
                // If hash of the block does not equal to chain hash
                if (!hash.equals(chainHash)) {

                    // Return error message
                    return "Hash of the last Block (Block " + i + ") does
not match with Chain Hash!";
                }
            }
        }
        // Return true if the blockchain is successfully verified
        return "TRUE";
    }

    /***
     * This routine repairs the chain. It checks the hashes of each block
and ensures
     * that any illegal hashes are recomputed. After this routine is run,
the chain will
     * be valid. The routine does not modify any difficulty values. It
computes new proof
```

```
     * of work based on the difficulty specified in the Block.
     */
    public void repairChain() {

        // Loop over every block in the chain
        for (int i = 0; i < ds_chain.size(); i++) {

            // Prepare String whose hash is to be found
            String hash_input = ds_chain.get(i).getIndex() +
ds_chain.get(i).getTimestamp().toString()
                    + ds_chain.get(i).getData() +
ds_chain.get(i).getPreviousHash()
                    + ds_chain.get(i).getNonce() +
ds_chain.get(i).getDifficulty();

            // Compute SHA 256 hash of the hash_input
            String hash = computeSHA256(hash_input);

            // Calculate leading zeros required to be present in the hash
            // Source to repeat a String multiple times:
            // https://stackoverflow.com/questions/1235179/simple-way-to-
repeat-a-string
            String leading_zeros_required = new String(new
char[ds_chain.get(i).getDifficulty()]).replace("\0", "0");

            // If the chain has only one Block
            if (ds_chain.size() == 1) {

                // If the previous hash is not ""
                if (!ds_chain.get(i).getPreviousHash().equals("")) {

                    // Set previous hash of the first node to be ""
                    ds_chain.get(i).setPreviousHash("");
                }
            }

            // If the hash does not start with the required number of
leading zeros
            if (!hash.startsWith(leading_zeros_required)) {

                // Computing new proof of work
                // Set initial nonce of the Block to be 0
                ds_chain.get(i).setNonce(BigInteger.valueOf(0));

                // Until the good hash is not found
                while (true)
                {
                    // Prepare new hash input String
                    String new_hash_input = ds_chain.get(i).getIndex()
                            + ds_chain.get(i).getTimestamp().toString()
                            + ds_chain.get(i).getData()
                            + ds_chain.get(i).getPreviousHash()
                            + ds_chain.get(i).getNonce()
                            + ds_chain.get(i).getDifficulty();

                    // Compute new hash
                    hash = computeSHA256(new_hash_input);

                    // If the new hash starts with the required number of
leading zeros
                    if (hash.startsWith(leading_zeros_required))
```

```
                    {
                        // End the while loop
                        break;
                    }

                    // If the new hash does not start with the required
number of leading zeros,
                    // increment the nonce by 1.

ds_chain.get(i).setNonce(ds_chain.get(i).getNonce().add(BigInteger.valueOf(
1)));
                }
            }

            // Prepare hash input after the above updates
            hash_input = ds_chain.get(i).getIndex() +
ds_chain.get(i).getTimestamp().toString()
                    + ds_chain.get(i).getData() +
ds_chain.get(i).getPreviousHash()
                    + ds_chain.get(i).getNonce() +
ds_chain.get(i).getDifficulty();

            // Compute SHA256 hash
            hash = computeSHA256(hash_input);

            // If the chain size is greater than 1
            if (ds_chain.size() > 1) {

                // If the block is not the last block
                if (i != ds_chain.size() - 1) {

                    // If the hash of the current block is not equal to
previous hash of the next block
                    if (!hash.equals(ds_chain.get(i +
1).getPreviousHash())) {

                        // Update the previous hash of the next block
                        ds_chain.get(i + 1).setPreviousHash(hash);
                    }
                }
                // If the block is the last block
                else {

                    // If the hash of the last block is not equal to chain
hash
                    if (!hash.equals(chainHash)) {

                        // Update chain hash
                        chainHash = hash;
                    }
                }
            }

            // If only one block is present in the chain
            else {

                // If hash of the block is not equal to chain hash
                if (!hash.equals(chainHash)) {

                    // Update chain hash
                    chainHash = hash;
```

```
                }
            }
        }
    }

    /***
     * This method uses the toString method defined on each individual
block
     * to convert the BlockChain object to a JSON string
     * @return A String representation of the entire chain
     */
    public java.lang.String toString() {
        // Create a Gson object
        Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd
HH:mm:ss.SSS").create();

        // Serialize to JSON
        return gson.toJson(this);
    }

    /***
     * computes the SHA256 hash value of the string passed
     * into the function, converts the hash into hexadecimal and
     * return the hash value
     * @param input String input whose hash values are to be computed
     * @return SHA256 hash value of the input string in hexadecimal form
     */
    public String computeSHA256(String input) {

        String hash = "";
        // Source: CMU 95702 Fall 2022 Lab1-InstallationAndRaft Code
        try {
            // Access MessageDigest class for SHA256
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            // Compute the digest
            md.update(input.getBytes());

            // Store digest as a byte array for further use
            byte[] digest = md.digest();

            hash = bytesToHex(digest);
        }
        // Handles No SHA-256 Algorithm exceptions
        catch (NoSuchAlgorithmException e) {
            // Print error message in console
            System.out.println("No SHA-256 available" + e);
        }

        return hash;
    }

    // Code to convert from byte array to hexadecimal String
    // Source: https://stackoverflow.com/questions/9655181/how-to-convert-
a-byte-array-to-a-hex-string-in-java
    private static final char[] HEX_ARRAY =
"0123456789ABCDEF".toCharArray();

    /***
     * Function to convert a byte array to hexadecimal String
     * @param bytes Byte array to be converted to hexadecimal String
```

```
     * @return Hexadecimal notation (in String form) of the input byte
array
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
        }
        return new String(hexChars);
    }


    public static void main(String[] args) {

        // Create a new object of the BlockChain
        BlockChain blockChain = new BlockChain();

        // Create the first Block, called the genesis Block
        Block genesis = new Block(0, blockChain.getTime(), "Genesis", 2);

        // Set the previous hash of the genesis block to be an empty String
        genesis.setPreviousHash("");

        // Compute the hashes per second on this system
        blockChain.computeHashesPerSecond();

        // Update chain hash by the hash of the genesis Block
        blockChain.chainHash = genesis.proofOfWork();

        // add Genesis block to chain
        blockChain.ds_chain.add(genesis);

        // Prompt to the user
        System.out.println("0. View basic blockchain status.\n" +
                "1. Add a transaction to the blockchain.\n" +
                "2. Verify the blockchain.\n" +
                "3. View the blockchain.\n" +
                "4. Corrupt the chain.\n" +
                "5. Hide the corruption by repairing the chain.\n" +
                "6. Exit");

        // Create a Scanner object
        Scanner s = new Scanner(System.in);

        // Get first input from the user
        int user_input = s.nextInt();

        // Until the user does not input 6
        while (user_input != 6) {

            // If user requested to view the status of the basic BlockChain
            if (user_input == 0) {

                // Displays the necessary details
                System.out.println("Current size of chain: " +
blockChain.getChainSize());
                System.out.println("Difficulty of most recent block: " +
blockChain.getLatestBlock().getDifficulty());
                System.out.println("Total difficulty for all blocks: " +
```

```
blockChain.getTotalDifficulty());
                System.out.println("Approximate hashes per second on this
machine: " + blockChain.getHashesPerSecond());
                System.out.println("Expected total hashes required for the
whole chain: " + String.format("%.6f",
blockChain.getTotalExpectedHashes()));
                System.out.println("Nonce for most recent block: " +
blockChain.getLatestBlock().getNonce());
                System.out.println("Chain hash: " +
blockChain.getChainHash());
            }

            // If user requested to add a transaction to the BlockChain
            else if (user_input == 1) {

                // Request for difficulty of the new Block
                System.out.println("Enter difficulty > 0");
                int difficulty = s.nextInt();

                // Request for transaction of the new Block
                System.out.println("Enter transaction");
                s.nextLine();
                String transaction = s.nextLine();

                // Create new Block
                Block newBlock = new Block(blockChain.getChainSize(),
blockChain.getTime(), transaction, difficulty);

                // Set previous hash of the new Block to be the chain hash
                newBlock.setPreviousHash(blockChain.getChainHash());

                // Source to calculate the run time of a program:
                // https://stackoverflow.com/questions/5204051/how-to-
calculate-the-running-time-of-my-program
                // Start time counter
                long startTime = System.nanoTime();

                // Add Block to BlockChain
                blockChain.addBlock(newBlock);

                // End time counter
                long endTime = System.nanoTime();

                // Calculate total time
                long totalTime = endTime - startTime;

                // Source to convert time in nanoseconds to milliseconds:
                // https://stackoverflow.com/questions/4300653/conversion-
of-nanoseconds-to-milliseconds-and-nanoseconds-999999-in-java
                long totalTimeInMilliSeconds =
TimeUnit.MILLISECONDS.convert(totalTime, TimeUnit.NANOSECONDS);

                // Display time required to add Block to the user
                System.out.println("Total execution time to add this block
was " + totalTimeInMilliSeconds + " milliseconds");
            }

            // If user requested to verify the Blockchain
            else if (user_input == 2) {

                // Source to calculate the run time of a program:
```

```
                // https://stackoverflow.com/questions/5204051/how-to-
calculate-the-running-time-of-my-program
                // Start time counter
                long startTime = System.nanoTime();

                // Compute chain verification result
                String chainVerificationResult = blockChain.isChainValid();

                // End time counter
                long endTime = System.nanoTime();

                // Compute total time required
                long totalTime = endTime - startTime;

                // Source to convert time in nanoseconds to milliseconds:
                // https://stackoverflow.com/questions/4300653/conversion-
of-nanoseconds-to-milliseconds-and-nanoseconds-999999-in-java
                long totalTimeInMilliSeconds =
TimeUnit.MILLISECONDS.convert(totalTime, TimeUnit.NANOSECONDS);

                // If the result of chain verification is true
                if (chainVerificationResult.equals("TRUE")) {

                    // Display success message to user
                    System.out.println("Chain verification: " +
chainVerificationResult);
                }

                // If the result of chain verification is false
                else {

                    // Display error message to user
                    System.out.println("Chain verification: FALSE");
                    System.out.println(chainVerificationResult);
                }

                // Display time required to verify the chain to the user
                System.out.println("Total execution time to verify the
chain was " + totalTimeInMilliSeconds + " milliseconds");
            }

            // If user requested to view the BlockChain
            else if (user_input == 3) {

                System.out.println("View the Blockchain");

                // Display BlockChain in JSON format
                System.out.println(blockChain);
            }

            // If user requested to corrupt the BlockChain
            else if (user_input == 4) {

                System.out.println("Corrupt the Blockchain");

                // Get ID of Block to corrupt
                System.out.println("Enter block ID of block to corrupt");
                int blockID = s.nextInt();

                // Enter new data of Block
                System.out.println("Enter new data for block " + blockID);
```

```
                s.nextLine();
                String newData = s.nextLine();

                // Update new data of Block in the chain
                blockChain.getBlock(blockID).setData(newData);

                // Update user about the corruption
                System.out.println("Block " + blockID + " now holds " +
blockChain.getBlock(blockID).getData());
            }

            // If user requested to hide the corruption in the BlockChain
by repairing the chain
            else if (user_input == 5) {

                // Source to calculate the run time of a program:
                // https://stackoverflow.com/questions/5204051/how-to-
calculate-the-running-time-of-my-program
                // Start time counter
                long startTime = System.nanoTime();

                // Repair block chain
                blockChain.repairChain();

                // End time counter
                long endTime = System.nanoTime();

                // Compute total time required to repair the chain
                long totalTime = endTime - startTime;

                // Source to convert time in nanoseconds to milliseconds:
                // https://stackoverflow.com/questions/4300653/conversion-
of-nanoseconds-to-milliseconds-and-nanoseconds-999999-in-java
                long totalTimeInMilliSeconds =
TimeUnit.MILLISECONDS.convert(totalTime, TimeUnit.NANOSECONDS);

                // Display time required to repair the chain to the user
                System.out.println("Total execution time required to repair
the chain was " + totalTimeInMilliSeconds + " milliseconds");
            }

            // Prompt to user for next input
            System.out.println("0. View basic blockchain status.\n" +
                    "1. Add a transaction to the blockchain.\n" +
                    "2. Verify the blockchain.\n" +
                    "3. View the blockchain.\n" +
                    "4. Corrupt the chain.\n" +
                    "5. Hide the corruption by repairing the chain.\n" +
                    "6. Exit");

            // Get next input from user
            user_input = s.nextInt();
        }


        /*
        Timing Data and Analysis

        - As the difficulty of the new blocks increases, the time to add
blocks (addBlock()) significantly increases. I see an exponential increase
in the approximate time taken to add blocks with increasing difficulty.
```

However, the time to verify the chain (isChainValid()) remains almost the same, no matter how large is the maximum difficulty in the blockchain. Contrary to verifying the chain, repairing the chain (chainRepair()) takes an increasingly large amount of time with the increasing difficulties.

1. Time Analysis for addBlock() with increasing difficulties

- Approximate run times for addBlock() with increasing difficulties. As the difficulties of the newly added blocks increases, the time taken to add the blocks to the blockchain increases exponentially as well. This is because, finding the proof of work and the good hash gets too hard for the system. With just a difficulty of 7, the expected hashes to be performed becomes 268,435,456. This is huge! And hence, the time taken to add the blocks is huge as well. For a difficulty of 8, the expected hashes count is 4,294,967,296; however, my computer is not capable to perform so many hashes and hence it crashes.

Difficulty	Approximate Time
1	2 milliseconds
2	5 milliseconds
3	27 milliseconds
4	85 milliseconds
5	1174 milliseconds
6	30947 milliseconds
7	111800 milliseconds
8	My system crashes! The program got hung. So, I did not try further.

2. Time Analysis for isChainValid() with increasing difficulties

- Approximate run time for isChainValid() with the following max difficulties in the blockchain. The time taken to validate a chain remains almost the same with increasing difficulties from 1 to 7. This is because, we do not have to compute the proof of work here with the increasing difficulties, rather, we just need to use it to find one hash and compare that with the difficulty. This is a lot simpler and hence validating chain does not take a lot of time even with the increasing difficulties.

Max Difficulty	Approximate Time
1	0 milliseconds
2	0 milliseconds
3	0 milliseconds
4	0 milliseconds
5	0 milliseconds
6	0 milliseconds
7	0 milliseconds
8	My system crashes! The program got hung. So, I did not try further.

3. Time Analysis for chainRepair() with increasing difficulties

- Approximate run time for chainRepair() when blocks with the following maximum difficulties were present in the blockchain and the blockchain was corrupted and repaired. As the maximum difficulty in the blockchain increases, the time to repair the chain (after it gets corrupted) increases significantly as well. As with the addBlock(), I see an exponential increase in time here as well. It is in line with addBlock(), where, to repair the chain, the program will have to compute

```
the proof of work and find the good hash. It requires computing a lot more
hashes (as described in the analysis of addBlock()) and thus add to the
exponential time.

        Maximum Difficulty     Approximate Time
        1                      1 milliseconds
        2                      3 milliseconds
        3                      12 milliseconds
        4                      17 milliseconds
        5                      694 milliseconds
        6                      32097 milliseconds
        7                      200187 milliseconds
        8                      My system crashes! The program got hung.
                               So, I did not try further.

        -  After analysing the approximate run times, it is clear for the
fact mentioned in class, “We give easy problems to the good guys and hard
problems to the bad guys.” The easy problems are solved quickly and the
hard problems take a lot of time to get solved.

         */
    }
}
```

Project 3 – Task 1

Task 1 Client Side Execution

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 2000000
Expected total hashes required for the whole chain: 256.000000
Nonce for most recent block: 16
Chain hash: 00CF74F079329C342461641E7975E0FC412CCF82A35E8C02E46057135580B2CF
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
2
Enter transaction
Alice pays Bob 100 DSCoin
Total execution time to add this block was 6 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
2
Enter transaction
Bob pays Carol 50 DSCoin
Total execution time to add this block was 4 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.

2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
2
Enter transaction
Carol pays Donna 10 DS Coin
Total execution time to add this block was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Chain verification: TRUE
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index":0,"timestamp":"2022-11-01
23:34:40.423","data":"Genesis","previousHash":"","nonce":16,"difficulty":2},{"index":1,"timestamp"
:"2022-11-01 23:35:10.498","data":"Alice pays Bob 100
DSCoin","previousHash":"00CF74F079329C342461641E7975E0FC412CCF82A35E8C02E46057135580
B2CF","nonce":189,"difficulty":2},{"index":2,"timestamp":"2022-11-01 23:35:36.256","data":"Bob
pays Carol 50
DSCoin","previousHash":"00B02E4E423A3964EFF296D3AE95326B2E71C96A50EA87BA5D503D0B4F
C41435","nonce":169,"difficulty":2},{"index":3,"timestamp":"2022-11-01
23:35:58.542","data":"Carol pays Donna 10 DS
Coin","previousHash":"005918FF79FEE33AD92D8D96514F8C36B6CC3C4DBF0AA34F20499B8404BC8
98E","nonce":50,"difficulty":2}],"chainHash":"008F533EF203191E239415F2EA9A9CEF2EBEF18BF5EE
527A0D57A4DCDC871BAB"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.

6. Exit
4
Corrupt the Blockchain
Enter block ID of block to corrupt
1
Enter new data for block 1
Alice pays Bob 76 DSCoin
Block 1 now holds Alice pays Bob 76 DSCoin
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index":0,"timestamp":"2022-11-01
23:34:40.423","data":"Genesis","previousHash":"","nonce":16,"difficulty":2},{"index":1,"timestamp"
:"2022-11-01 23:35:10.498","data":"Alice pays Bob 76
DSCoin","previousHash":"00CF74F079329C342461641E7975E0FC412CCF82A35E8C02E46057135580
B2CF","nonce":189,"difficulty":2},{"index":2,"timestamp":"2022-11-01 23:35:36.256","data":"Bob
pays Carol 50
DSCoin","previousHash":"00B02E4E423A3964EFF296D3AE95326B2E71C96A50EA87BA5D503D0B4F
C41435","nonce":169,"difficulty":2},{"index":3,"timestamp":"2022-11-01
23:35:58.542","data":"Carol pays Donna 10 DS
Coin","previousHash":"005918FF79FEE33AD92D8D96514F8C36B6CC3C4DBF0AA34F20499B8404BC8
98E","nonce":50,"difficulty":2}],"chainHash":"008F533EF203191E239415F2EA9A9CEF2EBEF18BF5EE
527A0D57A4DCDC871BAB"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Chain verification: FALSE
Improper hash on node 1. Does not begin with 00
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
5

Total execution time required to repair the chain was 6 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Chain verification: TRUE
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
4
Enter transaction
Donna pays Sean 25 DSCoin
Total execution time to add this block was 45 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
0
Current size of chain: 5
Difficulty of most recent block: 4
Total difficulty for all blocks: 12
Approximate hashes per second on this machine: 2000000
Expected total hashes required for the whole chain: 66560.000000
Nonce for most recent block: 14122
Chain hash: 000003B9DC0188E310D045DAE8F6D6DCFD2CB8EA55A8D88BBC1B69DFA0178438
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
6

Process finished with exit code 0

Task 1 Server Side Execution

Blockchain server running
We have a visitor
Response :
{"size":1,"totalHashes":256,"totalDiff":2,"recentNonce":16,"diff":2,"hps":2000000,"chainHash":"00CF74F079329C342461641E7975E0FC412CCF82A35E8C02E46057135580B2CF","selection":0}
Adding a block
Setting response to Total execution time to add this block was 6 milliseconds
...{"response":"Total execution time to add this block was 6 milliseconds","selection":1}
Adding a block
Setting response to Total execution time to add this block was 4 milliseconds
...{"response":"Total execution time to add this block was 4 milliseconds","selection":1}
Adding a block
Setting response to Total execution time to add this block was 0 milliseconds
...{"response":"Total execution time to add this block was 0 milliseconds","selection":1}
Verifying entire chain
Chain verification: TRUE
Total execution time required to verify the chain was 0
Setting response to Total execution time to verify the chain was 0 milliseconds
View the Blockchain
Setting response to {"ds_chain":[{"index":0,"timestamp":"2022-11-01 23:34:40.423","data":"Genesis","previousHash":"","nonce":16,"difficulty":2},{"index":1,"timestamp":"2022-11-01 23:35:10.498","data":"Alice pays Bob 100 DSCoin","previousHash":"00CF74F079329C342461641E7975E0FC412CCF82A35E8C02E46057135580B2CF","nonce":189,"difficulty":2},{"index":2,"timestamp":"2022-11-01 23:35:36.256","data":"Bob pays Carol 50 DSCoin","previousHash":"00B02E4E423A3964EFF296D3AE95326B2E71C96A50EA87BA5D503D0B4FC41435","nonce":169,"difficulty":2},{"index":3,"timestamp":"2022-11-01 23:35:58.542","data":"Carol pays Donna 10 DS Coin","previousHash":"005918FF79FEE33AD92D8D96514F8C36B6CC3C4DBF0AA34F20499B8404BC898E","nonce":50,"difficulty":2}],"chainHash":"008F533EF203191E239415F2EA9A9CEF2EBEF18BF5EE527A0D57A4DCDC871BAB"}
Corrupt the Blockchain
Block 1 now holds Alice pays Bob 76 DSCoin
Setting response to Block 1 now holds Alice pays Bob 76 DSCoin
View the Blockchain
Setting response to {"ds_chain":[{"index":0,"timestamp":"2022-11-01 23:34:40.423","data":"Genesis","previousHash":"","nonce":16,"difficulty":2},{"index":1,"timestamp":"2022-11-01 23:35:10.498","data":"Alice pays Bob 76 DSCoin","previousHash":"00CF74F079329C342461641E7975E0FC412CCF82A35E8C02E46057135580B2CF","nonce":189,"difficulty":2},{"index":2,"timestamp":"2022-11-01 23:35:36.256","data":"Bob pays Carol 50 DSCoin","previousHash":"00B02E4E423A3964EFF296D3AE95326B2E71C96A50EA87BA5D503D0B4FC41435","nonce":169,"difficulty":2},{"index":3,"timestamp":"2022-11-01 23:35:58.542","data":"Carol pays Donna 10 DS Coin","previousHash":"005918FF79FEE33AD92D8D96514F8C36B6CC3C4DBF0AA34F20499B8404BC8

98E","nonce":50,"difficulty":2}],"chainHash":"008F533EF203191E239415F2EA9A9CEF2EBEF18BF5EE527A0D57A4DCDC871BAB"}
Verifying entire chain
Chain verification: FALSE
Improper hash on node 1. Does not begin with 00
Total execution time required to verify the chain was 0
Setting response to Total execution time to verify the chain was 0 milliseconds
Repairing the entire chain
Setting response to Total execution time required to repair the chain was 6 milliseconds
Verifying entire chain
Chain verification: TRUE
Total execution time required to verify the chain was 0
Setting response to Total execution time to verify the chain was 0 milliseconds
Adding a block
Setting response to Total execution time to add this block was 45 milliseconds
...{"response":"Total execution time to add this block was 45 milliseconds","selection":1}
Response :
{"size":5,"totalHashes":66560,"totalDiff":12,"recentNonce":14122,"diff":4,"hps":2000000,"chainHash":"000003B9DC0188E310D045DAE8F6D6DCFD2CB8EA55A8D88BBC1B69DFA0178438","selection":0}

Task 1 Client Source Code

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Email: shpatel@cmu.edu
 * Last Modified: November 1, 2022
 * File: ClientTCP.java
 * Part Of: Project3Task1
 *
 * This Java file acts as the TCP Client who will make requests for
blockchain
 * operations over a TCP socket to the TCP Server. The client will be
focused
 * on driving the menu-driven interaction and communicating with the server
on
 * the backend. If the client exits, the server will still handle new
requests
 * with the existing blockchain intact.
 */

// Defines the package for the Java file
package org.example;

// imports required for TCP/IP, IO operations and Gson
import com.google.gson.Gson;
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class ClientTCP {

    public static void main(String[] args) throws IOException {

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Stores the input from the user
        String user_input;

        // Create a BufferReader object to get input from the user
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));

        // Keeping executing until the client is terminated
        while (true) {

            // Initialize user input to null
            user_input = "";

            // Prompt the user for operation
            System.out.println("0. View basic blockchain status.\n" +
                    "1. Add a transaction to the blockchain.\n" +
                    "2. Verify the blockchain.\n" +
                    "3. View the blockchain.\n" +
                    "4. Corrupt the chain.\n" +
                    "5. Hide the corruption by repairing the chain.\n" +
                    "6. Exit");

            // Update user input
```

```
            user_input = user_input + typed.readLine();

            // Stores the request message that would be sent to the server
            RequestMessage requestMessage;

            // Stores difficulty of block that the user will input
            int difficulty;

            // Stores blockId of block that the user will input
            int blockID;

            // Stores transaction data of block that the user will input
            String transactionData;

            // Create a Gson object
            Gson gson = new Gson();

            // Switch case for user input
            switch (user_input) {

                // If user requested to view the blockchain status
                case "0" -> {

                    // Create a normal request message
                    requestMessage = new NormalRequestMessage(0);

                    // Request the blockchain operation from server and
store the value of response
                    String serverReturned =
blockchain_operations(requestMessage.toString());

                    // Parse JSON response from server into
StatusResponseMessage
                    StatusResponseMessage responseMessage =
gson.fromJson(serverReturned, StatusResponseMessage.class);

                    // Display details to the user
                    System.out.println("Current size of chain: " +
responseMessage.size);
                    System.out.println("Difficulty of most recent block: "
+ responseMessage.diff);
                    System.out.println("Total difficulty for all blocks: "
+ responseMessage.totalDiff);
                    System.out.println("Approximate hashes per second on
this machine: " + responseMessage.hps);
                    System.out.println("Expected total hashes required for
the whole chain: " + String.format("%.6f", (double)
responseMessage.totalHashes));
                    System.out.println("Nonce for most recent block: " +
responseMessage.recentNonce);
                    System.out.println("Chain hash: " +
responseMessage.chainHash);
                }

                // If user requested to add a transaction to the blockchain
                case "1" -> {

                    // Prompt the user for difficulty
                    System.out.println("Enter difficulty > 0");
                    difficulty = Integer.parseInt(typed.readLine());
```

```
                    // Prompt the user for transaction
                    System.out.println("Enter transaction");
                    transactionData = typed.readLine();

                    // Create an add request message
                    requestMessage = new AddRequestMessage(1, difficulty,
transactionData);

                    // Request the blockchain operation from server and
store the value of response
                    String serverReturned =
blockchain_operations(requestMessage.toString());

                    // Parse JSON response from server into
NormalResponseMessage
                    NormalResponseMessage responseMessage =
gson.fromJson(serverReturned, NormalResponseMessage.class);

                    // Display response to the user
                    System.out.println(responseMessage.response);
                }

                // If user requested to verify the blockchain
                case "2" -> {

                    // Create a normal request message
                    requestMessage = new NormalRequestMessage(2);

                    // Request the blockchain operation from server and
store the value of response
                    String serverReturned =
blockchain_operations(requestMessage.toString());

                    // Parse JSON response from server into
VerificationResponseMessage
                    VerificationResponseMessage verificationResponseMessage
= gson.fromJson(serverReturned, VerificationResponseMessage.class);

                    // If chain verification was successful
                    if
(verificationResponseMessage.verificationOutput.equals("TRUE")) {
                        // Display success message
                        System.out.println("Chain verification: " +
verificationResponseMessage.verificationOutput);
                    }
                    // If chain verification was not successful
                    else {
                        // Display error message
                        System.out.println("Chain verification: FALSE");

System.out.println(verificationResponseMessage.verificationOutput);
                    }

                    // Display response to the user

System.out.println(verificationResponseMessage.response);
                }

                // If user requested to view the blockchain
                case "3" -> {
```

```
                    System.out.println("View the Blockchain");

                    // Create a normal request message
                    requestMessage = new NormalRequestMessage(3);

                    // Request the blockchain operation from server and
store the value of response
                    String serverReturned =
blockchain_operations(requestMessage.toString());

                    // Server would return a JSON message here
                    System.out.println(serverReturned);
                }

                // If user requested to corrupt the blockchain
                case "4" -> {

                    System.out.println("Corrupt the Blockchain");

                    // Prompt the user for difficulty
                    System.out.println("Enter block ID of block to
corrupt");
                    blockID = Integer.parseInt(typed.readLine());

                    // Prompt the user for transaction
                    System.out.println("Enter new data for block " +
blockID);
                    transactionData = typed.readLine();

                    // Create a corrupt request message
                    requestMessage = new CorruptRequestMessage(4, blockID,
transactionData);

                    // Request the blockchain operation from server and
store the value of response
                    String serverReturned =
blockchain_operations(requestMessage.toString());

                    // Parse JSON response from server into
NormalResponseMessage
                    NormalResponseMessage responseMessage =
gson.fromJson(serverReturned, NormalResponseMessage.class);

                    // Display response to the user
                    System.out.println(responseMessage.response);
                }

                // If user requested to hide the corruption by repairing
the chain
                case "5" -> {

                    // Create a normal request message
                    requestMessage = new NormalRequestMessage(5);

                    // Request the blockchain operation from server and
store the value of response
                    String serverReturned =
blockchain_operations(requestMessage.toString());

                    // Parse JSON response from server into
NormalResponseMessage
```

```
                    NormalResponseMessage responseMessage =
gson.fromJson(serverReturned, NormalResponseMessage.class);

                    // Display response to the user
                    System.out.println(responseMessage.response);
                }

                // If user requested to exit
                case "6" -> {

                    // Halt client execution
                    System.exit(0);
                }
            }
        }
    }

    /***
     * Function to communicate with the server and perform the required
     * operation on the requested integer value by the client
     * @param user_input Input from the user containing client ID,
operation and operand
     * @return Updated sum from the server
     */

    /***
     * Function to communicate with the server and perform the required
     * blockchain operation. The request message from the client would be
     * in the form of a JSON string.
     * @param jsonRequestMessage Request message from client
     * @return JSON response from server
     */
    public static String blockchain_operations(String jsonRequestMessage) {

        // Define a TCP style Socket
        Socket clientSocket = null;

        // Stores the JSON response from the server
        String serverReturned = "";

        try {
            // Creating a String object for localhost
            String localhost = "";

            // Updating clientSocket with localhost and the server port
            clientSocket = new Socket(localhost, 6789);

            // Set up "in" to read from the server socket
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            // Set up "out" to write to the server socket
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));

            // Request to the server with the JSON request message
            out.println(jsonRequestMessage);

            // Flush to server socket
            out.flush();
```

```
            // Store the JSON response from the server
            serverReturned = in.readLine(); // read a line of data from the
stream
        }
        // Handle general I/O exceptions
        catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        }
        // Always close the socket
        finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }

        // Return the JSON response from server
        return serverReturned;
    }
}
```

Task 1 Server Source Code

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Email: shpatel@cmu.edu
 * Last Modified: November 1, 2022
 * File: ServerTCP.java
 * Part Of: Project3Task1
 *
 * This Java file acts as the TCP Server who will get requests for the TCP
Client
 * in the form on JSON strings and perform the necessary blockchain
operations.
 * The blockchain will exist on the server. It will be constructed there
and the
 * client will make requests for operations over a TCP socket. If the
client exits,
 * the server will still handle new requests with the existing blockchain
intact.
 */

// Defines the package for the Java file
package org.example;

// imports required for TCP/IP, IO operations and Gson
import com.google.gson.Gson;
import java.net.*;
import java.io.*;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;

public class ServerTCP {

    // Stores an array list of blocks in the blockchain
    static BlockChain blockChain = new BlockChain();

    // Stores the JSON response to be sent to the client
    static String json_response = null;

    // Stores the ResponseMessage to be converted to JSON which would be
later sent to the client
    static ResponseMessage responseMessage;

    // Stores the response which is a part of the overall response message
    static String response;

    // Create a Gson object
    static Gson gson = new Gson();

    public static void main(String[] args) {

        // Define a TCP style Socket
        Socket clientSocket = null;

        // Define a TCP style ServerSocket
        ServerSocket listenSocket;
        try {

            // Hard coded port for the server (as suggest on Piazza)
```

```
            int serverPort = 6789;

            // Create a new server socket
            listenSocket = new ServerSocket(serverPort);

            // Create the first Block, called the genesis Block
            Block genesis = new Block(0, blockChain.getTime(), "Genesis",
2);

            // Set the previous hash of the genesis block to be an empty
String
            genesis.setPreviousHash("");

            // Compute the hashes per second on this system
            blockChain.computeHashesPerSecond();

            // Update chain hash by the hash of the genesis Block
            blockChain.chainHash = genesis.proofOfWork();

            // add Genesis block to chain
            blockChain.ds_chain.add(genesis);

            System.out.println("Blockchain server running");
            System.out.println("We have a visitor");

            /*
             * Forever,
             *   read a line from the socket
             *   print it to the console
             *   echo it (i.e. write it) back to the client
             */
            while (true) {
                /*
                 * Block waiting for a new connection request from a
client.
                 * When the request is received, "accept" it, and the rest
                 * the tcp protocol handshake will then take place, making
                 * the socket ready for reading and writing.
                 */
                clientSocket = listenSocket.accept();
                // If we get here, then we are now connected to a client.

                // Set up "in" to read from the client socket
                Scanner in;
                in = new Scanner(clientSocket.getInputStream());

                // Set up "out" to write to the client socket
                PrintWriter out;
                out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));

                // Get the input from the client in JSON format
                String user_input = in.nextLine();

                // Convert JSON client request into a RequestMessage format
                RequestMessage requestMessage = gson.fromJson(user_input,
RequestMessage.class);

                // If user requested to view the status of the basic
BlockChain
                if (requestMessage.operation == 0) {
```

```
                    // Form the JSON response by calling
viewBlockChainStatus()
                    json_response =
viewBlockChainStatus(requestMessage.operation);

                    // Display response to user
                    System.out.println("Response : " + json_response);
                }

                // If user requested to add a transaction to the BlockChain
                else if (requestMessage.operation == 1) {
                    System.out.println("Adding a block");

                    // Form a AddRequestMessage from the client request
                    AddRequestMessage addRequestMessage =
gson.fromJson(user_input, AddRequestMessage.class);

                    // Form the JSON response by calling addTransaction()
                    json_response = addTransaction(addRequestMessage);

                    // Display response to user
                    System.out.println("..." + json_response);
                }

                // If user requested to verify the Blockchain
                else if (requestMessage.operation == 2) {
                    System.out.println("Verifying entire chain");

                    // Form a NormalRequestMessage from the client request
                    NormalRequestMessage normalRequestMessage =
gson.fromJson(user_input, NormalRequestMessage.class);

                    // Form the JSON response by calling verifyBlockChain()
                    json_response = verifyBlockChain(normalRequestMessage);
                }

                // If user requested to view the BlockChain
                else if (requestMessage.operation == 3) {
                    System.out.println("View the Blockchain");

                    // Form the JSON response by calling viewBlockChain()
                    json_response = viewBlockChain();

                    // Display response to user
                    System.out.println("Setting response to " +
json_response);
                }

                // If user requested to corrupt the BlockChain
                else if (requestMessage.operation == 4) {
                    System.out.println("Corrupt the Blockchain");

                    // Form a CorruptRequestMessage from the client request
                    CorruptRequestMessage corruptRequestMessage =
gson.fromJson(user_input, CorruptRequestMessage.class);

                    // Form the JSON response by calling
corruptBlockChain()
                    json_response =
corruptBlockChain(corruptRequestMessage);
```

```
                    // Display response to user
                    System.out.println("Setting response to " + response);
                }

                // If user requested to hide the corruption in the
BlockChain by repairing the chain
                else if (requestMessage.operation == 5) {
                    System.out.println("Repairing the entire chain");

                    // Form a NormalRequestMessage from the client request
                    NormalRequestMessage normalRequestMessage =
gson.fromJson(user_input, NormalRequestMessage.class);

                    // Form the JSON response by calling repairBlockChain()
                    json_response = repairBlockChain(normalRequestMessage);

                    // Display response to user
                    System.out.println("Setting response to " + response);
                }

                // Reply the JSON response to the client
                out.println(json_response);

                // Flush to client socket
                out.flush();
            }
        }
        // Handle IO exceptions
        catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
            // If quitting (typically by you sending quit signal) clean up
sockets
        }
        // Always close the socket
        finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    /***
     * Function to view the current blockchain status
     * @param requestMessage_operation requested operation number from the
client
     * @return JSON response
     */
    public static String viewBlockChainStatus(int requestMessage_operation)
{

        // Create a StatusResponseMessage object
        ResponseMessage responseMessage = new StatusResponseMessage(
                requestMessage_operation,
                blockChain.getChainSize(),
                blockChain.getChainHash(),
                (int) blockChain.getTotalExpectedHashes(),
```

```
                blockChain.getTotalDifficulty(),
                blockChain.getLatestBlock().getNonce(),
                blockChain.getLatestBlock().getDifficulty(),
                blockChain.getHashesPerSecond());

        // Convert the object to JSON
        json_response = gson.toJson(responseMessage);

        // JSON response to client
        return json_response;
    }

    /***
     * Function to add block to the blockchain
     * @param addRequestMessage AddRequestMessage from client
     * @return JSON response
     */
    public static String addTransaction(AddRequestMessage
addRequestMessage) {

        // Create new Block
        Block newBlock = new Block(blockChain.getChainSize(),
blockChain.getTime(),
                addRequestMessage.transactionData,
addRequestMessage.difficulty);

        // Set previous hash of the new Block to be the chain hash
        newBlock.setPreviousHash(blockChain.getChainHash());

        // Source to calculate the run time of a program:
        // https://stackoverflow.com/questions/5204051/how-to-calculate-
the-running-time-of-my-program
        // Start time counter
        long startTime = System.nanoTime();

        // Add Block to BlockChain
        blockChain.addBlock(newBlock);

        // End time counter
        long endTime = System.nanoTime();

        // Calculate total time
        long totalTime = endTime - startTime;

        // Source to convert time in nanoseconds to milliseconds:
        // https://stackoverflow.com/questions/4300653/conversion-of-
nanoseconds-to-milliseconds-and-nanoseconds-999999-in-java
        long totalTimeInMilliSeconds =
TimeUnit.MILLISECONDS.convert(totalTime, TimeUnit.NANOSECONDS);

        // Create response string
        response = "Total execution time to add this block was " +
totalTimeInMilliSeconds + " milliseconds";

        System.out.println("Setting response to " + response);

        // Create a NormalResponseMessage
        responseMessage = new
NormalResponseMessage(addRequestMessage.operation, response);

        // Convert the object to JSON
```

```
        json_response = gson.toJson(responseMessage);

        // JSON response to client
        return json_response;
    }

    /***
     * Function to verify blockchain
     * @param normalRequestMessage NormalRequestMessage from client
     * @return JSON response
     */
    public static String verifyBlockChain(NormalRequestMessage
normalRequestMessage) {
        // Source to calculate the run time of a program:
        // https://stackoverflow.com/questions/5204051/how-to-calculate-
the-running-time-of-my-program
        // Start time counter
        long startTime = System.nanoTime();

        // Compute chain verification result
        String chainVerificationResult = blockChain.isChainValid();

        // End time counter
        long endTime = System.nanoTime();

        // Compute total time required
        long totalTime = endTime - startTime;

        // Source to convert time in nanoseconds to milliseconds:
        // https://stackoverflow.com/questions/4300653/conversion-of-
nanoseconds-to-milliseconds-and-nanoseconds-999999-in-java
        long totalTimeInMilliSeconds =
TimeUnit.MILLISECONDS.convert(totalTime, TimeUnit.NANOSECONDS);

        // If the result of chain verification is true
        if (chainVerificationResult.equals("TRUE")) {

            // Display success message to user
            System.out.println("Chain verification: " +
chainVerificationResult);
        }

        // If the result of chain verification is false
        else {

            // Display error message to user
            System.out.println("Chain verification: FALSE");
            System.out.println(chainVerificationResult);
        }

        // Define response message
        response = "Total execution time to verify the chain was " +
totalTimeInMilliSeconds + " milliseconds";

        // Display time required to verify to user
        System.out.println("Total execution time required to verify the
chain was " + totalTimeInMilliSeconds);

        // Display response to user
        System.out.println("Setting response to " + response);
```

```
        // Create a VerificationResponseMessage
        responseMessage = new
VerificationResponseMessage(normalRequestMessage.operation, response,
chainVerificationResult);

        // Convert the object to JSON
        json_response = gson.toJson(responseMessage);

        // JSON response to client
        return json_response;
    }

    /***
     * Function to view blockchain
     * @return JSON response
     */
    public static String viewBlockChain() {

        // Convert blockCahin object to JSON string format
        json_response = blockChain.toString(); // would be a json message

        // JSON response to client
        return json_response;
    }

    /***
     * Function to corrupt blockchain
     * @param corruptRequestMessage CorruptRequestMessage from client
     * @return JSON response
     */
    public static String corruptBlockChain(CorruptRequestMessage
corruptRequestMessage) {

        // Stores block ID of the Block to corrupt
        int blockID = corruptRequestMessage.blockID;

        // Stores corrupted data to be stored in the Block
        String newData = corruptRequestMessage.transactionData;

        // Corrupt block
        blockChain.getBlock(blockID).setData(newData);

        // Define response message
        response = "Block " + blockID + " now holds " +
blockChain.getBlock(blockID).getData();

        // Display response to user
        System.out.println(response);

        // Create a NormalResponseMessage
        responseMessage = new
NormalResponseMessage(corruptRequestMessage.operation, response);

        // Convert the object to JSON
        json_response = gson.toJson(responseMessage);

        // JSON response to client
        return json_response;
    }

    /***
```

```
     * Function to repaid blockchain
     * @param normalRequestMessage NormalRequestMessage from client
     * @return JSON response
     */
    public static String repairBlockChain(NormalRequestMessage
normalRequestMessage) {
        // Source to calculate the run time of a program:
        // https://stackoverflow.com/questions/5204051/how-to-calculate-
the-running-time-of-my-program
        // Start time counter
        long startTime = System.nanoTime();

        // Repair block chain
        blockChain.repairChain();

        // End time counter
        long endTime = System.nanoTime();

        // Compute total time required to repair the chain
        long totalTime = endTime - startTime;

        // Source to convert time in nanoseconds to milliseconds:
        // https://stackoverflow.com/questions/4300653/conversion-of-
nanoseconds-to-milliseconds-and-nanoseconds-999999-in-java
        long totalTimeInMilliSeconds =
TimeUnit.MILLISECONDS.convert(totalTime, TimeUnit.NANOSECONDS);

        // Define response message
        response = "Total execution time required to repair the chain was "
+ totalTimeInMilliSeconds + " milliseconds";

        // Create a NormalResponseMessage
        responseMessage = new
NormalResponseMessage(normalRequestMessage.operation, response);

        // Convert the object to JSON
        json_response = gson.toJson(responseMessage);

        // JSON response to client
        return json_response;
    }
}
```