95-702 Distributed Systems
Project 3

Author: Jennifer Chen (Yu Chen)
Andrew Id: yuc3 (yuc3@andrew.cmu.edu)
Date: 3/15/2022

Task 0 Execution (below is the copy and paste from the console, I also put the screenshot up here for your reference)

```
/Users/chenyu/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55214:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/chenyu/IdeaProjects/Project3Task0/out/production/Project3Task0 BlockChain
Hello! Welcome to my Project3 task0!
-----------------------------------------

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: 1376462
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 769
```

Chain hash: 003697EC06D7A78F8D4EA2028C8132AEA49DFED702130DAD193762DA887A8EF7

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 100DSCoin
Total execution time to add this block was 37 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 22 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 Andy 10DSCoin
Total execution time to add this block 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 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

3
View the Blockchain:
{"ds_chain" : [{"index": 0, "time stamp": "2022-03-16 00:09:45.61", "Tx": "Genesis", "PrevHash": "", "nonce": 769, "difficulty": 2},
{"index": 1, "time stamp": "2022-03-16 00:10:50.701", "Tx": "Alice pays Bob 100DSCoin", "PrevHash":
"003697EC06D7A78F8D4EA2028C8132AEA49DFED702130DAD193762DA887A8EF7", "nonce": 302, "difficulty": 2},
{"index": 2, "time stamp": "2022-03-16 00:12:08.397", "Tx": "Bob pays Carol 50 DSCoin", "PrevHash":
"0093B6D934D69DB59E01A8ACEE3756935B1825E9D0C513DED4C40A60A3223887", "nonce": 774, "difficulty": 2},
{"index": 3, "time stamp": "2022-03-16 00:13:08.367", "Tx": "Carol pays Andy 10DSCoin", "PrevHash":
"00CBFC8EA475C7CF7A7CACFEE3296EB9E6966DB5B958E9F5E39DF07F748E2995", "nonce": 628, "difficulty": 2}
], "chainHash":"00F4BE7B82ADAB2A3F7AEA0C44D58B84BB25D1A173DD54608898D82BDA548657"}

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, "time stamp": "2022-03-16 00:09:45.61", "Tx": "Genesis", "PrevHash": "", "nonce": 769, "difficulty": 2},
{"index": 1, "time stamp": "2022-03-16 00:10:50.701", "Tx": "Alice pays Bob 76 DSCoin", "PrevHash":
"003697EC06D7A78F8D4EA2028C8132AEA49DFED702130DAD193762DA887A8EF7", "nonce": 302, "difficulty": 2},
{"index": 2, "time stamp": "2022-03-16 00:12:08.397", "Tx": "Bob pays Carol 50 DSCoin", "PrevHash":
"0093B6D934D69DB59E01A8ACEE3756935B1825E9D0C513DED4C40A60A3223887", "nonce": 774, "difficulty": 2},
{"index": 3, "time stamp": "2022-03-16 00:13:08.367", "Tx": "Carol pays Andy 10DSCoin", "PrevHash":
"00CBFC8EA475C7CF7A7CACFEE3296EB9E6966DB5B958E9F5E39DF07F748E2995", "nonce": 628, "difficulty": 2}
], "chainHash":"00F4BE7B82ADAB2A3F7AEA0C44D58B84BB25D1A173DD54608898D82BDA548657"}

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 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

5
Total execution time required to repair the chain was 17 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 11 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:
Andy pays Sean 25 DSCoin
Total execution time to add this block was 80 milliseconds

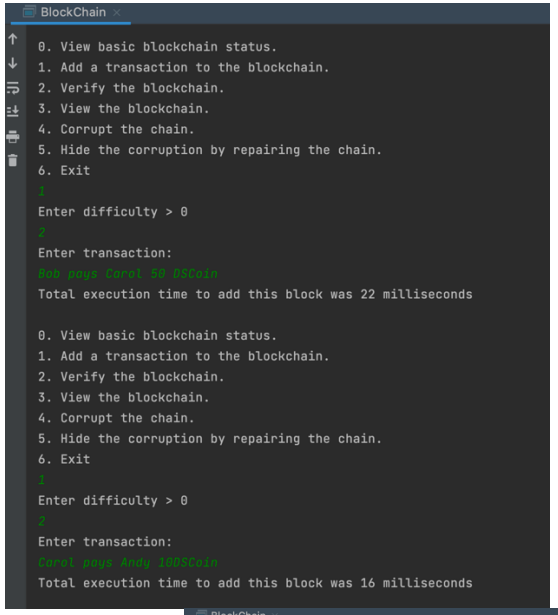
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

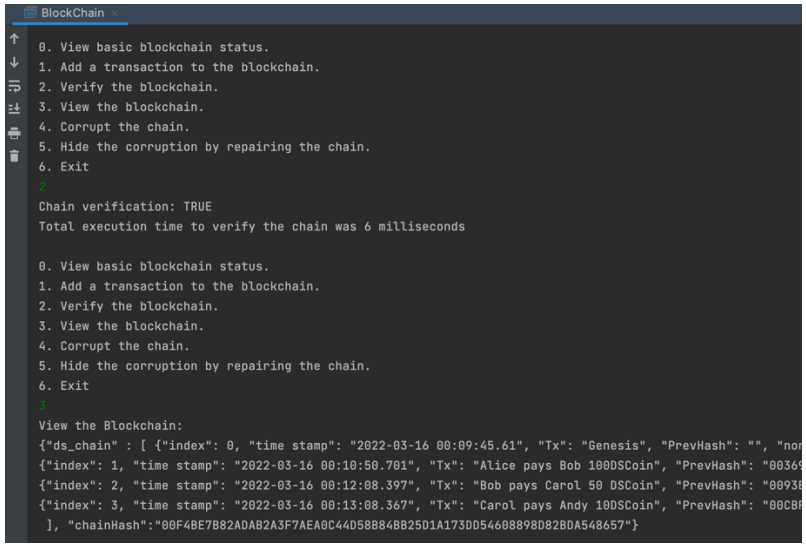
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
0
Current size of chain:5
Difficulty of most recent block: 4
Total difficulty for all blocks: 12
Approximate hashes per second on this machine: 1376462
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 18728
Chain hash: 0000B2BC52D9CC889A5C49CA9371C61FAB54A71EE0F888BE93AC339DF9EE062C

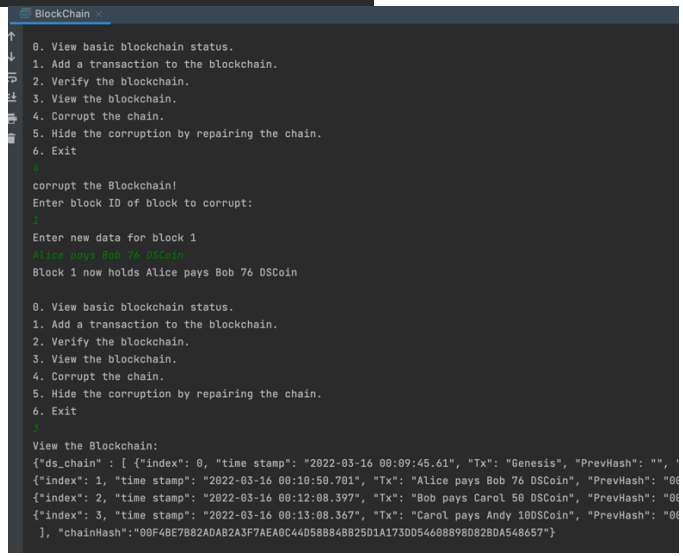
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
6
Bye Bye!

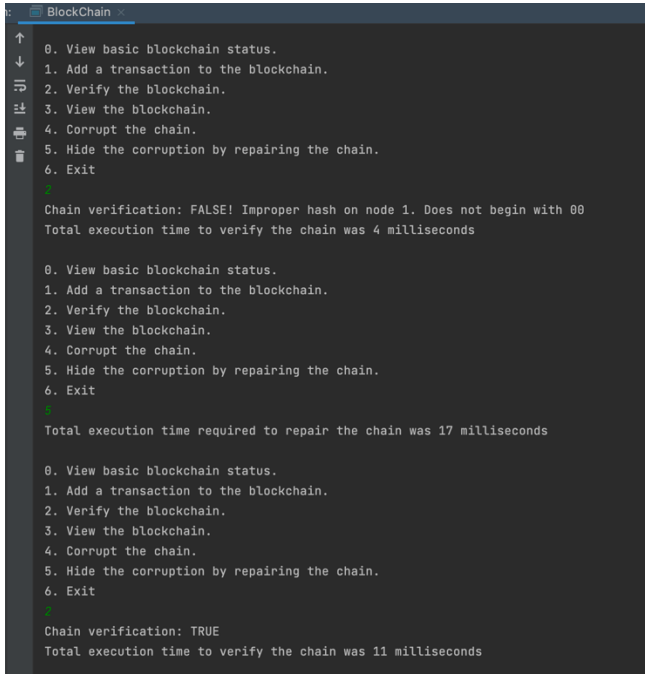
Process finished with exit code 0

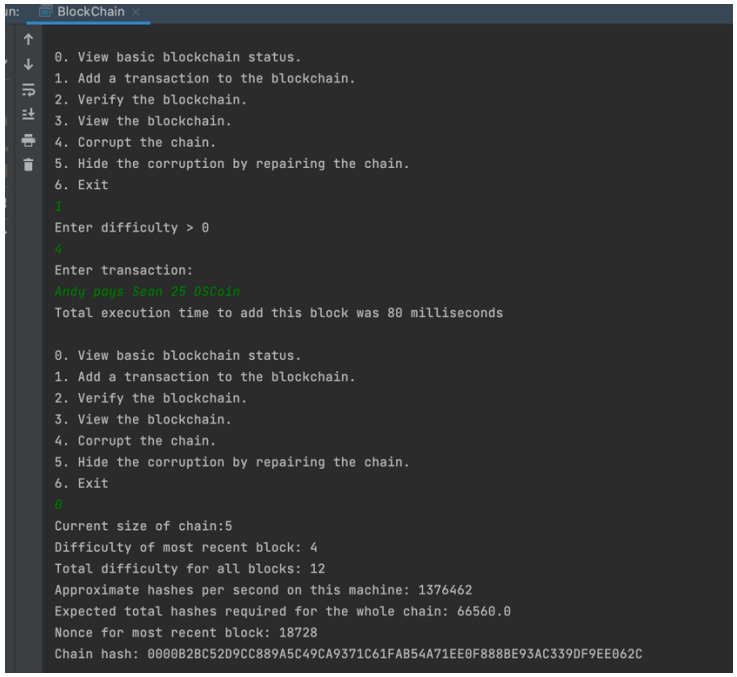

BlockChain
/Users/chenyu/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java -
Hello! Welcome to my Project3 task0!
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption 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: 1376462
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 769
Chain hash: 003697EC06D7A78F8D4EA2028C8132AEA49DFED702130DAD193762DA887A8EF7
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 100DSCoin
Total execution time to add this block was 37 milliseconds


BlockChain
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 22 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 Andy 10DSCoin
Total execution time to add this block was 16 milliseconds


BlockChain
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 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
3
View the Blockchain:
{"ds_chain" : [{"index": 0, "time stamp": "2022-03-16 00:09:45.61", "Tx": "Genesis", "PrevHash": "", "nor
{"index": 1, "time stamp": "2022-03-16 00:10:50.701", "Tx": "Alice pays Bob 100DSCoin", "PrevHash": "00369
{"index": 2, "time stamp": "2022-03-16 00:12:08.397", "Tx": "Bob pays Carol 50 DSCoin", "PrevHash": "00938
{"index": 3, "time stamp": "2022-03-16 00:13:08.367", "Tx": "Carol pays Andy 10DSCoin", "PrevHash": "00CB8
], "chainHash":"00F4BE7B82ADAB2A3F7AEA0C44D58B84BB25D1A173DD54608898D82BDA548657"}


BlockChain
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, "time stamp": "2022-03-16 00:09:45.61", "Tx": "Genesis", "PrevHash": "",
{"index": 1, "time stamp": "2022-03-16 00:10:50.701", "Tx": "Alice pays Bob 76 DSCoin", "PrevHash": "00
{"index": 2, "time stamp": "2022-03-16 00:12:08.397", "Tx": "Bob pays Carol 50 DSCoin", "PrevHash": "00
{"index": 3, "time stamp": "2022-03-16 00:13:08.367", "Tx": "Carol pays Andy 10DSCoin", "PrevHash": "00
], "chainHash":"00F4BE7B82ADAB2A3F7AEA0C44D58B84BB25D1A173DD54608898D82BDA548657"}


BlockChain
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 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
5
Total execution time required to repair the chain was 17 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 11 milliseconds


BlockChain
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:
Andy pays Sean 25 DSCoin
Total execution time to add this block was 80 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: 1376462
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 18728
Chain hash: 0000B2BC52D9CC889A5C49CA9371C61FAB54A71EE0F888BE93AC339DF9EE062C


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
Bye Bye!
Process finished with exit code 0

Task 0 Block.java

```java
//Name: Yu Chen
//Andrew Id: yuc3
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

/**
 * This is the Project 3 for 95-702.
 * The reference doc is: https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/Block.html
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class Block {
    /**
     * Instance variable - the position of the block on the chain.
     */
    private int index;
    /**
     * Instance variable - holds the time of the block's creation.
     */
    private Timestamp timestamp;
    /**
     * Instance variable - holding the block's single transaction details.
     */
    private String data;
    /**
     * Instance variable - the minimum number of left most hex digits needed by a proper hash.
     */
    private int difficulty;
    /**
     * Instance variable - determined by a proof of work routine.
     */
    private BigInteger nonce;
    /**
     * Instance variable - the SHA256 hash of a block's parent(hash pointer).
     */
    private String previousHash;
```

```java
    /**
     * This the Block constructor.
     * @param index - This is the position within the chain. Genesis is at 0.
     * @param timestamp - This is the time this block was added.
     * @param data - This is the transaction to be included on the blockchain.
     * @param difficulty - This is the number of leftmost nibbles that need to be 0.
     */
    public Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.difficulty = difficulty;
    }

    /**
     * This method returns the nonce for this block.
     * The nonce is a number that has been found to cause
     * the hash of this block to have the correct number of leading hexadecimal zeroes.
     *
     * @return a BigInteger representing the nonce for this block
     */
    public BigInteger getNonce() {
        return nonce;
    }

    /**
     * Simple getter method.
     *
     * @return difficulty
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * @return data
     */
    public String getData() {
        return data;
    }
```

```java
    /**
     * @return index
     */
    public int getIndex() {
        return index;
    }

    /**
     * @return previous hash
     */
    public String getPreviousHash() {
        return previousHash;
    }

    /**
     * @return timestamp
     */
    public Timestamp getTimestamp() {
        return timestamp;
    }

    /**
     * Simple setter method
     * @param difficulty - determines how much work is required to produce a proper hash
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Simple setter method
     * @param previousHash - a hashpointer to this block's parent
     */
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    /**
     * Simple setter method
     * @param index - the index of this block in the chain
     */
```

```
    public void setIndex(int index) {
        this.index = index;

    }

    /**
     * Simple setter method
     * @param data - represents the transaction held by this block
     */
    public void setData(String data) {
        this.data = data;

    }

    /**
     * Simple setter method
     * @param timestamp - of when this block was created
     */
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }

    /**
     * This method computes a hash of the concatenation of the index, timestamp, data, previousHash, nonce,
and difficulty.
     *
     * @return a String holding Hexadecimal characters
     */
    public String calculateHash() throws NoSuchAlgorithmException {
        StringBuilder sb = new StringBuilder();

sb.append(index).append(timestamp).append(data).append(previousHash).append(nonce).append(difficulty);
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(sb.toString().getBytes());
        return bytesToHex(md.digest());
    }

    // Reference from: 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();
```

```
    public 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);
    }

    /**
     * The proof of work methods finds 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.
     */
    public String proofOfWork() throws NoSuchAlgorithmException {
        BigInteger n = new BigInteger("0");
        String leadingZeros = "0".repeat(Math.max(0, difficulty));
        String s = calculateHash();

        while (!s.substring(0, difficulty).equals(leadingZeros)) {
            n = n.add(new BigInteger("1"));
            nonce = n; //set the nonce
            s = calculateHash();
        }

        return calculateHash();
    }


    /**
     * @return A JSON representation of all of this block's data is returned.
```

```
     */
    @Override
    public String toString() { //output to a json format
        return "{\"index\": "+ getIndex() +", \"time stamp\": \""+ getTimestamp() +
                "\", \"Tx\": \""+ getData() +"\", \"PrevHash\": \""+ getPreviousHash() +
                "\", \"nonce\": "+ getNonce() + ", \"difficulty\": "+ getDifficulty() +
                "}";
    }

    public static void main(String[] args){

    }
}
```

Task 0 BlockChain.java

```
//Name: Yu Chen
//Andrew Id: yuc3

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.sql.Timestamp;

/**
 * This is the Project 3 for 95-702.
 * The reference doc is: https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html
 *
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class BlockChain {
    /**
     * Instance member - an ArrayList to hold Blocks.
     */
    private List<Block> blocks;
```

```java
    /**
     * Instance variable - a chain hash to hold a SHA256 hash of the most recently added Block.
     */
    private String chainHash;
    /**
     * Instance variable - approximate number of hashes per second on this computer.
     */
    private long hashesPerSecond;

    /**
     * Constructor - This BlockChain has exactly three instance members:
     * an ArrayList to hold Blocks and a chain hash to hold a SHA256 hash of the most recently added Block.
     */
    public BlockChain() {
        blocks = new ArrayList<>();
        chainHash = "";
        hashesPerSecond = 0;
    }

    /**
     * A new Block is being added to the BlockChain.
     * This new block's previous hash must hold the hash of the most recently added block.
     * After this call on addBlock, the new block becomes the most recently added block on the BlockChain.
     *
     * @param newBlock - is added to the BlockChain as the most recent block
     * @throws NoSuchAlgorithmException
     */
    public void addBlock(Block newBlock) throws NoSuchAlgorithmException {
        newBlock.setPreviousHash(chainHash);
        newBlock.proofOfWork();
        blocks.add(newBlock);
        chainHash = newBlock.calculateHash();
    }

    /**
     * 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() throws NoSuchAlgorithmException {
```

```
        long start = System.currentTimeMillis();
        int i = 0;
        while (i < 2000000) {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update("00000000".getBytes());
            bytesToHex(md.digest());
            i++;
        }
        long end = System.currentTimeMillis();
        hashesPerSecond = (long) ((long) 2000000 / ((end - start) / 1000.0));
    }

    // Reference from : 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();

    public 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);
    }

    /**
     * @param i
     * @return block at postion i
     */
    public Block getBlock(int i) {
        return blocks.get(i);
    }

    public String getChainHash() {
        return chainHash;
    }

    /**
     * @return the size of the chain in blocks.
     */
```

```
    public int getChainSize() {
        return blocks.size();
    }

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

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

    /**
     * @return the current system time.
     */
    public Timestamp getTime() {
        return new Timestamp(System.currentTimeMillis());
    }

    /**
     * Compute and return the total difficulty of all blocks on the chain. Each block knows its own
difficulty.
     *
     * @return total difficulty
     */
    public int getTotalDifficulty() {
        int res = 0;
        for (Block b : blocks) {
            res += b.getDifficulty();
        }
        return res;
    }

    /**
     * Compute and return the expected number of hashes required for the entire chain.
```

```
     *
     * @return getTotalExpectedHashes
     */
    public double getTotalExpectedHashes() {
        double res = 0;
        for (Block b : blocks) {
            int blockPerHash = 1;
            int times = b.getDifficulty();
            for (int i = 0; i < times; i++) {
                blockPerHash *= 16;
            }
            res += blockPerHash;
        }
        return res;
    }

    /**
     * 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".
     * <p>
     * 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
     * @throws NoSuchAlgorithmException
     */
    public String isChainValid() throws NoSuchAlgorithmException {
        if (blocks.size() == 1) { //if the chain only contains one block
            String hash = blocks.get(0).calculateHash();
            int hashZeros = 0;
            int index = 0;
            while (hash.charAt(index) == '0') {
                hashZeros++;
                index++;
```

```
            }
            if (!hash.equals(chainHash) && hashZeros != blocks.get(0).getDifficulty()) {
                return "FALSE! Improper hash on node 0. Does not begin with " +
"0".repeat(blocks.get(0).getDifficulty());
            }
        } else { //if the chain contains more then one block
            for (int i = 0; i < blocks.size(); i++) {
                String hash = blocks.get(i).calculateHash();
                int hashZeros = 0;
                int index = 0;
                while (hash.charAt(index) == '0') {
                    hashZeros++;
                    index++;
                }
                if (i != blocks.size() - 1 &&
                        !hash.equals(blocks.get(i + 1).getPreviousHash()) && hashZeros !=
blocks.get(i).getDifficulty()) {
                    return "FALSE! Improper hash on node " + i + ". " +
                            "Does not begin with " + "0".repeat(blocks.get(i).getDifficulty());
                }
                if (i == blocks.size() - 1 && // for the last block of the chain
                        !hash.equals(chainHash) && hashZeros != blocks.get(i).getDifficulty()) {
                    return "FALSE! Improper hash on node " + i + ". " +
                            "Does not begin with " + "0".repeat(blocks.get(i).getDifficulty());
                }
            }
        }
        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.
     *
     * @throws NoSuchAlgorithmException
     */
    public void repairChain() throws NoSuchAlgorithmException {
        if (blocks.size() == 1) { //if the chain only contains one block
```

```
            String hash = blocks.get(0).calculateHash();
            int hashZeros = 0;
            int index = 0;
            while (hash.charAt(index) == '0') {
                hashZeros++;
                index++;
            }
            if (!hash.equals(chainHash) && hashZeros != blocks.get(0).getDifficulty()) {
                blocks.get(0).proofOfWork(); //need to compute proofOfWork again
            }
        } else { //if the chain contains more then one block
            for (int i = 0; i < blocks.size(); i++) {
                String hash = blocks.get(i).calculateHash();
                int hashZeros = 0;
                int index = 0;
                while (hash.charAt(index) == '0') {
                    hashZeros++;
                    index++;
                }
                if (i != blocks.size() - 1 &&
                        !hash.equals(blocks.get(i + 1).getPreviousHash()) && hashZeros !=
blocks.get(i).getDifficulty()) {
                    String newHash = blocks.get(i).proofOfWork();
                    blocks.get(i + 1).setPreviousHash(newHash); //reset the hashes
                    blocks.get(i + 1).setTimestamp(getTime()); //reset the time stamp
                }
                if (i == blocks.size() - 1 &&
                        !hash.equals(chainHash) && hashZeros != blocks.get(i).getDifficulty()) {
                    blocks.get(i).proofOfWork();
                    chainHash = blocks.get(i).calculateHash();
                }
            }
        }


    }

    @Override
    public String toString() { //out put to a json format
        StringBuilder sb = new StringBuilder();
        sb.append("{\"ds_chain\" : [ ");
```

```
        for (int i = 0; i < blocks.size(); i++) {
            if (i != blocks.size() - 1) {
                sb.append(blocks.get(i)).append(",");
            } else {
                sb.append(blocks.get(i));
            }
            sb.append("\n"); //for better formatting (one line per block)
        }
        sb.append(" ], \"chainHash\":\"").append(chainHash).append("\"}");
        return sb.toString();
    }


    /**
     * This routine acts as a test driver for your Blockchain.
     * It will begin by creating a BlockChain object and then adding the Genesis block to the chain.
     * <p>
     * I write the requirement comment in the end of the main method!!!!!!
     *
     * @param args
     * @throws NoSuchAlgorithmException
     */
    public static void main(String[] args) throws NoSuchAlgorithmException {
        BlockChain bc = new BlockChain();
        //The Genesis block will be created with an empty string as the previous hash and a difficulty of 2
        Block genesis = new Block(0, bc.getTime(), "Genesis", 2);
        bc.addBlock(genesis);
        bc.computeHashesPerSecond(); //establish the hashes per second instance member
        Scanner input = new Scanner(System.in);
        //menu driven
        System.out.println("Hello! Welcome to my Project3 task0!");
        System.out.println("-------------------------------------------");
        System.out.println();

        int choice;
        while (true) {
            long start;
            long end;
            System.out.println("0. View basic blockchain status.");
            System.out.println("1. Add a transaction to the blockchain.");
            System.out.println("2. Verify the blockchain.");
```

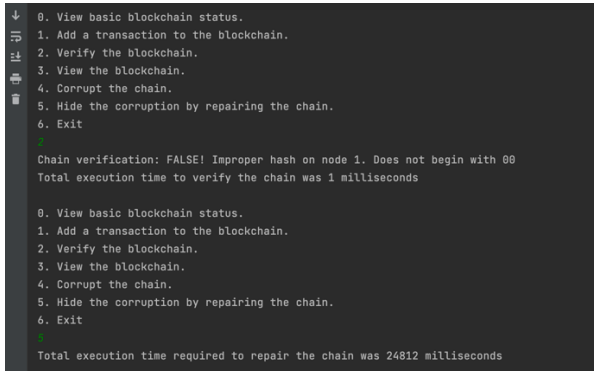
```
            System.out.println("3. View the blockchain.");
            System.out.println("4. Corrupt the chain.");
            System.out.println("5. Hide the corruption by repairing the chain.");
            System.out.println("6. Exit");
            choice = input.nextInt();

            switch (choice) {
                case 0:
                    System.out.println("Current size of chain:" + bc.getChainSize());
                    System.out.println("Difficulty of most recent block: " +
bc.getLatestBlock().getDifficulty());
                    System.out.println("Total difficulty for all blocks: " + bc.getTotalDifficulty());
                    System.out.println("Approximate hashes per second on this machine: " +
bc.getHashesPerSecond());
                    System.out.println("Expected total hashes required for the whole chain: " +
bc.getTotalExpectedHashes());
                    System.out.println("Nonce for most recent block: " + bc.getLatestBlock().getNonce());
                    System.out.println("Chain hash: " + bc.getChainHash());
                    System.out.println();
                    break;

                case 1:
                    input.nextLine();
                    System.out.println("Enter difficulty > 0");
                    int diff = input.nextInt();
                    System.out.println("Enter transaction: ");
                    input.nextLine();
                    String tx = input.nextLine();
                    Block b = new Block(bc.getChainSize(), bc.getTime(), tx, diff);
                    start = System.currentTimeMillis();
                    bc.addBlock(b);
                    end = System.currentTimeMillis();
                    long time = end - start;
                    System.out.println("Total execution time to add this block was " + time + "
milliseconds");
                    System.out.println();
                    break;

                case 2:
                    start = System.currentTimeMillis();
                    System.out.println("Chain verification: " + bc.isChainValid());
```

```
                    end = System.currentTimeMillis();
                    System.out.println("Total execution time to verify the chain was " + (end - start) + "
milliseconds");
                    System.out.println();
                    break;

                case 3:
                    System.out.println("View the Blockchain: ");
                    System.out.println(bc);
                    System.out.println();
                    break;

                case 4:
                    System.out.println("corrupt the Blockchain!");
                    System.out.println("Enter block ID of block to corrupt:");
                    int index = input.nextInt();
                    System.out.println("Enter new data for block " + index);
                    input.nextLine();
                    String newData = input.nextLine();
                    bc.blocks.get(index).setData(newData);
                    System.out.println("Block " + index + " now holds " + newData);
                    System.out.println();
                    break;

                case 5:
                    start = System.currentTimeMillis();
                    bc.repairChain();
                    end = System.currentTimeMillis();
                    System.out.println("Total execution time required to repair the chain was " + (end -
start) + " milliseconds");
                    System.out.println();
                    break;

                case 6:
                    System.out.println("Bye Bye!");
                    System.exit(0);
                    break;

                default:
                    System.out.println("Enter only 0-6 please!");
                    break;
```

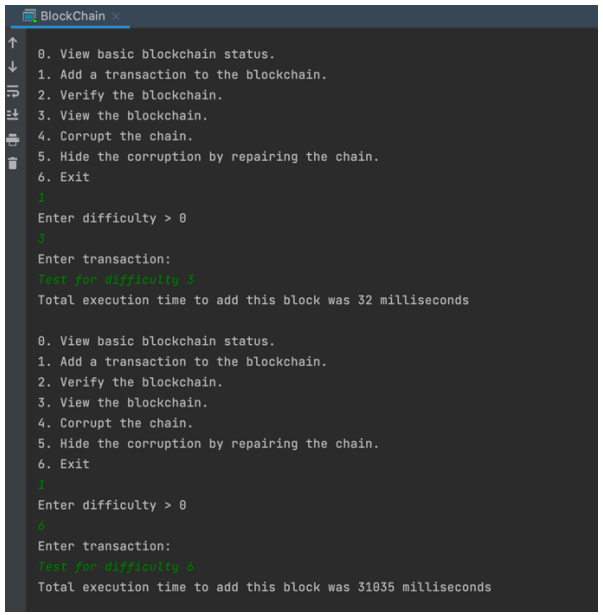
```
            }
        }

        /**
         * Within your comments in the main routine,
         * you must describe how this system behaves as the difficulty increases.
         * Run some experiments by adding new blocks with increasing difficulties.
         * Describe what you find. Be specific and quote some times.
         * You need not employ a system clock.
         * You should be able to make clear statements describing the approximate run times associated with
         * addBlock(), isChainValid(), and chainRepair().
         *
         * My analysis:
         *
         * [FOR addBlock()]
         * When the difficulty increases, the execution time to add block will also increase!
         * For example:
         * difficulty = 2 --> time = 43 milliseconds
         * difficulty = 3 --> time = 32 milliseconds
         *
         * //sometime might have some small error which do not increase,
         * but it will be obvious when the difficulty largely increase
         *
         * difficulty = 4 --> time = 68 milliseconds
         * difficulty = 5 --> time = 834 milliseconds
         * difficulty = 6 --> time = 31035 milliseconds
         *
         * [FOR isChainValid()]
         * isChainValid()'s execution time is always really quick.
         * For example:
         * 3 milliseconds, 0 milliseconds, 1 milliseconds......
         *
         * [FOR chainRepair()]
         * chainRepair()'s execution time will be very long since it need to calculate every block's proof
of work
         * after the corrupt index's block.
         * For example:
         * For a blockchain with current size of chain is 6 and total difficulty for all blocks is 22,
         * I corrupt index 1's data, so when I call
         * chainRepair(), it would need to re-calculate index 1, 2, 3, 4, 5's proof of work, which is
really
```

```
         * time consuming.
         * ex: Total execution time required to repair the chain was 24812 milliseconds
         *
         * CONCLUSION: Blockchains are easy to validate but time consuming to modify !
         *
         * //screenshot is in the pdf for your reference
         *
         */

    }


}
```

Some screenshots of my analysis when execute my code:



→ quick to validate but slow to modify



→ difficulty increases, the execution time to add block will also increase

---

Task 1 Client Side Execution

/Users/chenyu/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55722:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/chenyu/IdeaProjects/Project3Task1/target/classes:/Users/chenyu/.m2/repository/com/fasterxml/jackson/core/jackson-databind/2.9.8/jackson-databind-2.9.8.jar:/Users/chenyu/.m2/repository/com/fasterxml/jackson/core/jackson-annotations/2.9.0/jackson-annotations-2.9.0.jar:/Users/chenyu/.m2/repository/com/fasterxml/jackson/core/jackson-core/2.9.8/jackson-core-2.9.8.jar ClientTCP
The client is running.
Please input the server port number:
6789
Server port: 6789
==========
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: 1542020
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: null
Chain hash: 00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6

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 24 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 Andy 10 DSCoin
Total execution time to add this block was 36 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 this block was 8 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,
    "time stamp" : "2022-03-16 02:03:12.098",
```

```
   "Tx" : "Genesis",
   "PrevHash" : "",
   "nonce" : null,
   "difficulty" : 2
  }, {
   "index" : 1,
   "time stamp" : "2022-03-16 02:03:40.185",
   "Tx" : "Alice pays Bob 100 DSCoin",
   "PrevHash" : "00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6",
   "nonce" : 31,
   "difficulty" : 2
  }, {
   "index" : 2,
   "time stamp" : "2022-03-16 02:03:51.448",
   "Tx" : "Bob pays Carol 50 DSCoin",
   "PrevHash" : "00E8009DB98619EB73A2F0E277074A47A3F15648196039F5B41472F4905B3B40",
   "nonce" : 221,
   "difficulty" : 2
  }, {
   "index" : 3,
   "time stamp" : "2022-03-16 02:04:05.228",
   "Tx" : "Carol pays Andy 10 DSCoin",
   "PrevHash" : "005A8DD693BACDB4ECF11FA2A31494BEF2522D4125B51B9689AFB67845FA5E77",
   "nonce" : 437,
   "difficulty" : 2
  } ],
  "chainHash" : "0009E071722B18646E050EA5BAC987FD813E352B938CA88095F335B315AA621F"
}
```


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,
    "time stamp" : "2022-03-16 02:03:12.098",
    "Tx" : "Genesis",
    "PrevHash" : "",
    "nonce" : null,
    "difficulty" : 2
  }, {

```
   "index" : 1,
   "time stamp" : "2022-03-16 02:03:40.185",
   "Tx" : "Alice pays Bob 76 DSCoin",
   "PrevHash" : "00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6",
   "nonce" : 31,
   "difficulty" : 2
  }, {
   "index" : 2,
   "time stamp" : "2022-03-16 02:03:51.448",
   "Tx" : "Bob pays Carol 50 DSCoin",
   "PrevHash" : "00E8009DB98619EB73A2F0E277074A47A3F15648196039F5B41472F4905B3B40",
   "nonce" : 221,
   "difficulty" : 2
  }, {
   "index" : 3,
   "time stamp" : "2022-03-16 02:04:05.228",
   "Tx" : "Carol pays Andy 10 DSCoin",
   "PrevHash" : "005A8DD693BACDB4ECF11FA2A31494BEF2522D4125B51B9689AFB67845FA5E77",
   "nonce" : 437,
   "difficulty" : 2
  } ],
  "chainHash" : "0009E071722B18646E050EA5BAC987FD813E352B938CA88095F335B315AA621F"
}
```


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 this block was 23 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 to repair this block was 17 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 this block was 9 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:
Andy pays Sean 25 DSCoin
Total execution time to add this block was 11 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: 1542020
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 659
Chain hash: 000057A1B7A02802268FAB22E7E3B836403416BDDEAD9CC46EB7A621A3BCFE2C

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
Bye Bye!

Process finished with exit code 0

Task 1 Server Side Execution

/Users/chenyu/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55725:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/chenyu/IdeaProjects/Project3Task1/target/classes:/Users/chenyu/.m2/repository/com/fasterxml/jackson/core/jackson-databind/2.9.8/jackson-databind-2.9.8.jar:/Users/chenyu/.m2/repository/com/fasterxml/jackson/core/jackson-annotations/2.9.0/jackson-annotations-2.9.0.jar:/Users/chenyu/.m2/repository/com/fasterxml/jackson/core/jackson-core/2.9.8/jackson-core-2.9.8.jar ServerTCP
The server is running.
Please input the port number to listen on:
6789
Now listen on port: 6789
==========
Blockchain server running
Response:
{"selection":0,"size":1,"chainHash":"00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6","totalHashes":256.0,"totalDiff":2,"recentNonce":null,"diff":2,"hps":1542020}

Adding a block
Setting response to Total execution time to add this block was 6 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 6 milliseconds"}

Adding a block
Setting response to Total execution time to add this block was 24 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 24 milliseconds"}

Adding a block
Setting response to Total execution time to add this block was 36 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 36 milliseconds"}

Verifying entire chain
Chain verification: TRUE
Total execution time to verify this block was 8 milliseconds
Setting response to Total execution time to verify this block was 8 milliseconds

View the Blockchain
Setting response to {"ds_chain" : [{"index": 0, "time stamp": "2022-03-16 02:03:12.098", "Tx": "Genesis", "PrevHash": "", "nonce": null, "difficulty": 2},{"index": 1, "time stamp": "2022-03-16 02:03:40.185", "Tx": "Alice pays Bob 100 DSCoin", "PrevHash": "00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6", "nonce": 31, "difficulty": 2},{"index": 2, "time stamp": "2022-03-16 02:03:51.448", "Tx": "Bob pays Carol 50 DSCoin", "PrevHash": "00E8009DB98619EB73A2F0E277074A47A3F15648196039F5B41472F4905B3B40", "nonce": 221, "difficulty": 2},{"index": 3, "time stamp": "2022-03-16 02:04:05.228", "Tx": "Carol pays Andy 10 DSCoin", "PrevHash": "005A8DD693BACDB4ECF11FA2A31494BEF2522D4125B51B9689AFB67845FA5E77", "nonce": 437, "difficulty": 2}], "chainHash":"0009E071722B18646E050EA5BAC987FD813E352B938CA88095F335B315AA621F"}

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, "time stamp": "2022-03-16 02:03:12.098", "Tx": "Genesis", "PrevHash": "", "nonce": null, "difficulty": 2},{"index": 1, "time stamp": "2022-03-16 02:03:40.185", "Tx": "Alice pays Bob 76 DSCoin", "PrevHash": "00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6", "nonce": 31, "difficulty": 2},{"index": 2, "time stamp": "2022-03-16 02:03:51.448", "Tx": "Bob pays Carol 50 DSCoin", "PrevHash": "00E8009DB98619EB73A2F0E277074A47A3F15648196039F5B41472F4905B3B40", "nonce": 221, "difficulty": 2},{"index": 3, "time stamp": "2022-03-16 02:04:05.228", "Tx": "Carol pays Andy 10 DSCoin", "PrevHash":

"005A8DD693BACDB4ECF11FA2A31494BEF2522D4125B51B9689AFB67845FA5E77", "nonce": 437, "difficulty": 2}],
"chainHash":"0009E071722B18646E050EA5BAC987FD813E352B938CA88095F335B315AA621F"}

Verifying entire chain
Chain verification: FALSE! Improper hash on node 1. Does not begin with 00
Total execution time to verify this block was 23 milliseconds
Setting response to Total execution time to verify this block was 23 milliseconds

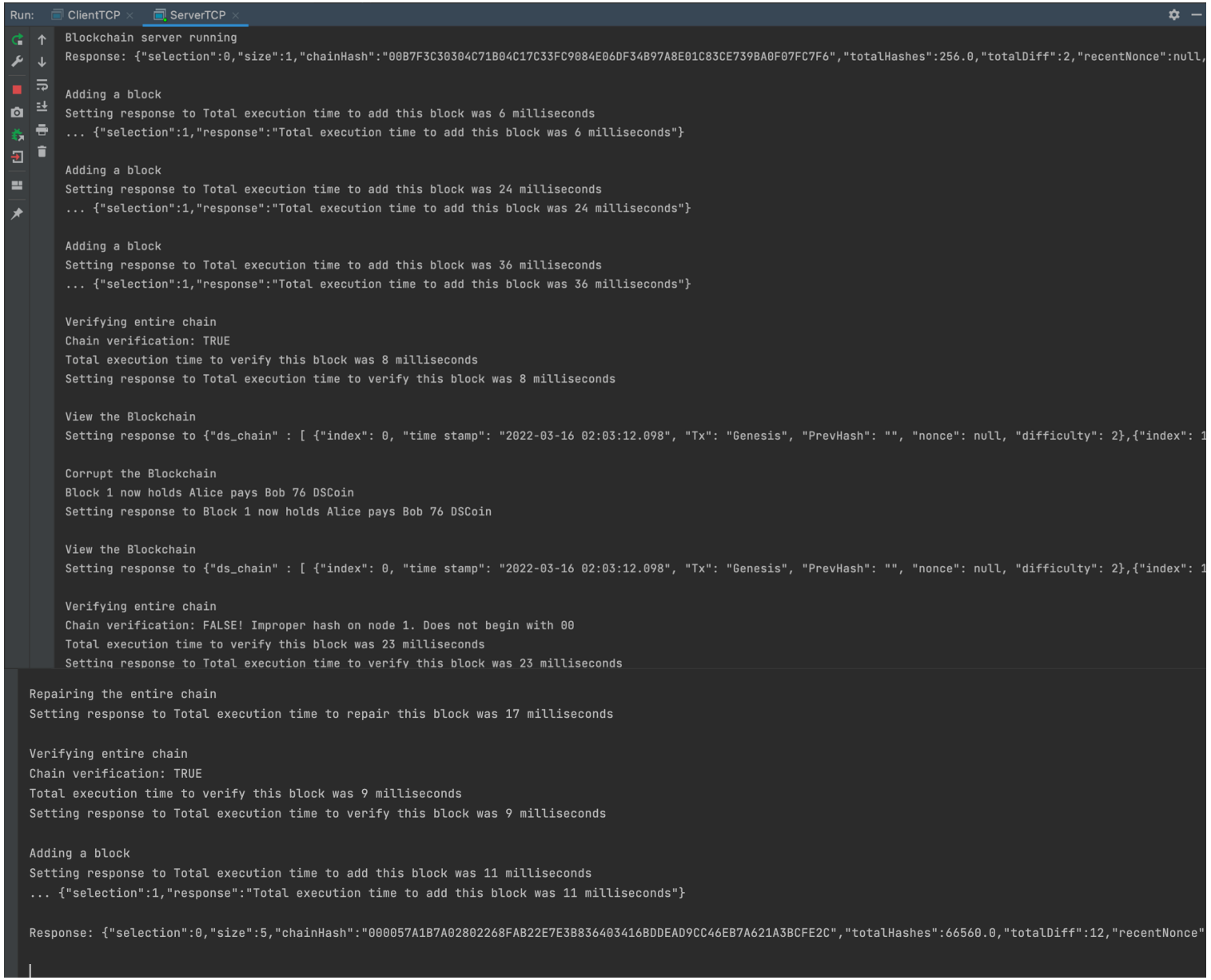
Repairing the entire chain
Setting response to Total execution time to repair this block was 17 milliseconds

Verifying entire chain
Chain verification: TRUE
Total execution time to verify this block was 9 milliseconds
Setting response to Total execution time to verify this block was 9 milliseconds

Adding a block
Setting response to Total execution time to add this block was 11 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 11 milliseconds"}

Response:
{"selection":0,"size":5,"chainHash":"000057A1B7A02802268FAB22E7E3B836403416BDDEAD9CC46EB7A621A3BCFE2C","totalHashes":66560.0,"totalDiff":12,"recentNonce":659,"diff":4,"hps":1542020}

screenshot of server-side execution for your reference:


Run:
ClientTCP
ServerTCP
Blockchain server running
Response: {"selection":0,"size":1,"chainHash":"00B7F3C30304C71B04C17C33FC9084E06DF34B97A8E01C83CE739BA0F07FC7F6","totalHashes":256.0,"totalDiff":2,"recentNonce":null,
Adding a block
Setting response to Total execution time to add this block was 6 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 6 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 24 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 24 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 36 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 36 milliseconds"}
Verifying entire chain
Chain verification: TRUE
Total execution time to verify this block was 8 milliseconds
Setting response to Total execution time to verify this block was 8 milliseconds
View the Blockchain
Setting response to {"ds_chain" : [{"index": 0, "time stamp": "2022-03-16 02:03:12.098", "Tx": "Genesis", "PrevHash": "", "nonce": null, "difficulty": 2},{"index": 1
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, "time stamp": "2022-03-16 02:03:12.098", "Tx": "Genesis", "PrevHash": "", "nonce": null, "difficulty": 2},{"index": 1
Verifying entire chain
Chain verification: FALSE! Improper hash on node 1. Does not begin with 00
Total execution time to verify this block was 23 milliseconds
Setting response to Total execution time to verify this block was 23 milliseconds
Repairing the entire chain
Setting response to Total execution time to repair this block was 17 milliseconds
Verifying entire chain
Chain verification: TRUE
Total execution time to verify this block was 9 milliseconds
Setting response to Total execution time to verify this block was 9 milliseconds
Adding a block
Setting response to Total execution time to add this block was 11 milliseconds
... {"selection":1,"response":"Total execution time to add this block was 11 milliseconds"}
Response: {"selection":0,"size":5,"chainHash":"000057A1B7A02802268FAB22E7E3B836403416BDDEAD9CC46EB7A621A3BCFE2C","totalHashes":66560.0,"totalDiff":12,"recentNonce"

Task 1 Client Source Code (ClientTCP.java)

```
//Name: Yu Chen
//Andrew Id: yuc3

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.*;
import java.net.Socket;
import java.util.Scanner;

/**
 * This is the Project 3 Task 1 for 95-702.
 *
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class ClientTCP {
    private static int serverPort;

    //some of this part I reused my code in Project2 Task4
    public static String toServer(String str) {

        Socket clientSocket = null;
        String res = "";
        try {
            clientSocket = new Socket("localhost", serverPort);

            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));

            res = in.readLine();

        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
```

```
            }
        } catch (IOException e) {
            // ignore exception on close
        }
    }

    return res;

}

public static void main(String args[]) throws IOException {
    RequestMessage rm = new RequestMessage();
    Scanner input = new Scanner(System.in);
    System.out.println("The client is running."); //declare that the client is running
    System.out.println("Please input the server port number: ");
    serverPort = input.nextInt(); // get user input as serverPort
    System.out.println("Server port: " + serverPort);
    System.out.println("=============");

    //I use Jackson as the Java JSON library
    //reference : https://www.baeldung.com/jackson-object-mapper-tutorial
    //reference : http://tutorials.jenkov.com/java-json/jackson-objectmapper.html

    int choice;
    while (true) {
        long start;
        long end;
        System.out.println("0. View basic blockchain status.");
        System.out.println("1. Add a transaction to the blockchain.");
        System.out.println("2. Verify the blockchain.");
        System.out.println("3. View the blockchain.");
        System.out.println("4. Corrupt the chain.");
        System.out.println("5. Hide the corruption by repairing the chain.");
        System.out.println("6. Exit");
        choice = input.nextInt();
        StringBuilder sb = new StringBuilder();
        String request = "";
        String response = "";
        ObjectMapper mapper = new ObjectMapper();
        JsonNode root;
```

```
            switch (choice) {
                case 0:
                    request = rm.toJsonFormat(0, "");
                    response = toServer(request);
                    root = mapper.readTree(response);

                    System.out.println("Current size of chain:" + root.get("size"));
                    System.out.println("Difficulty of most recent block: " + root.get("diff"));
                    System.out.println("Total difficulty for all blocks: " + root.get("totalDiff"));
                    System.out.println("Approximate hashes per second on this machine: " +
root.get("hps"));
                    System.out.println("Expected total hashes required for the whole chain: " +
root.get("totalHashes"));
                    System.out.println("Nonce for most recent block: " + root.get("recentNonce"));
                    System.out.println("Chain hash: " + root.get("chainHash").textValue());
                    System.out.println();
                    //System.out.println(response);
                    break;

                case 1:
                    input.nextLine();
                    System.out.println("Enter difficulty > 0");
                    int diff = input.nextInt();
                    System.out.println("Enter transaction: ");
                    input.nextLine();
                    String tx = input.nextLine();
                    sb.append(diff).append("#").append(tx);
                    request = rm.toJsonFormat(1, sb.toString());
                    //System.out.println(request);
                    response = toServer(request);
                    root = mapper.readTree(response);
                    System.out.println(root.get("response").textValue());
                    System.out.println();

                    break;

                case 2:
                    request = rm.toJsonFormat(2, "");
                    response = toServer(request);
                    root = mapper.readTree(response);
                    System.out.println("Chain verification: " + root.get("valid").textValue());
```

```
                System.out.println(root.get("response").textValue());
                System.out.println();
                break;

            case 3:
                request = rm.toJsonFormat(3, "");
                response = toServer(request);
                System.out.println("View the Blockchain: ");
                //for better read output
                root = mapper.readTree(response);
                String prettyFormat = mapper.writerWithDefaultPrettyPrinter().writeValueAsString(root);
                System.out.println(prettyFormat);
                System.out.println();
                break;

            case 4:
                System.out.println("corrupt the Blockchain!");
                System.out.println("Enter block ID of block to corrupt:");
                int index = input.nextInt();
                System.out.println("Enter new data for block " + index);
                input.nextLine();
                String newData = input.nextLine();
                sb.append(index).append("#").append(newData);
                request = rm.toJsonFormat(4, sb.toString());
                response = toServer(request);
                root = mapper.readTree(response);
                System.out.println(root.get("response").textValue());
                System.out.println();

                break;

            case 5:
                request = rm.toJsonFormat(5, "");
                response = toServer(request);
                root = mapper.readTree(response);
                System.out.println(root.get("response").textValue());

                System.out.println();
                break;

            case 6:
```

```
                    System.out.println("Bye Bye!");
                    System.exit(0);
                    break;

                default:
                    System.out.println("Enter only 0-6 please!");
                    break;
            }
        }
    }


}
```

Task 1 Client Source Code (RequestMessage.java)

```
//Name: Yu Chen
//Andrew Id: yuc3
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;

/**
 * This is the Project 3 Task 1 for 95-702.
 *
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class RequestMessage {

    /**
     * I use Jackson as the Java JSON library
     * reference 1 : https://www.baeldung.com/jackson-object-mapper-tutorial
     * reference 2 : http://tutorials.jenkov.com/java-json/jackson-objectmapper.html
     *
     * @param selection
     * @param data
     * @return json format request message to send to server
     * @throws JsonProcessingException
     */
    public String toJsonFormat(int selection, String data) throws JsonProcessingException {
```

```
        ObjectMapper mapper = new ObjectMapper();

        ObjectNode outerObject = mapper.createObjectNode();
        outerObject.putPOJO("selection", selection);
        String[] strs = data.split("#");

        switch (selection) {
            case 0:
            case 2:
            case 3:
            case 5:
                break; //if the user choice is 0 or 2 or 3 or 5, just send the selection field to server
            case 1:
                String tx = mapper.writeValueAsString(strs[1]);
                outerObject.putPOJO("difficulty", Integer.parseInt(strs[0]));
                outerObject.putPOJO("Tx", tx);
                break;
            case 4:
                String newTx = mapper.writeValueAsString(strs[1]);
                outerObject.putPOJO("index", Integer.parseInt(strs[0]));
                outerObject.putPOJO("newData", newTx);
                break;

        }

        return outerObject.toString();
    }
}
```

Task 1 Server Source Code (ServerTCP.java)

```
//Name: Yu Chen
//Andrew Id: yuc3

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```java
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

/**
 * This is the Project 3 Task 1 for 95-702.
 *
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class ServerTCP {
    //some of this part I reused my code in Project2 Task4
    //I use Jackson as the Java JSON library
    //reference : https://www.baeldung.com/jackson-object-mapper-tutorial
    //reference : http://tutorials.jenkov.com/java-json/jackson-objectmapper.html

    public static void main(String args[]) throws NoSuchAlgorithmException {
        BlockChain bc = new BlockChain();
        Block genesis = new Block(0, bc.getTime(), "Genesis", 2);
        bc.addBlock(genesis);
        bc.computeHashesPerSecond();

        System.out.println("The server is running.");
        Scanner input = new Scanner(System.in);
        System.out.println("Please input the port number to listen on: ");
        int serverPort = input.nextInt();
        System.out.println("Now listen on port: " + serverPort);
        System.out.println("============");

        System.out.println("Blockchain server running");

        Socket clientSocket = null;
        try {
            // Create a new server socket
            ServerSocket listenSocket = new ServerSocket(serverPort);

            /*
             * 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.
             */
            while (true) {
                clientSocket = listenSocket.accept();
                // If we get here, then we are now connected to a client.

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

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

                String data = in.readLine();

                ResponseMessage rm = new ResponseMessage();
                ObjectMapper mapper = new ObjectMapper();
                if (data != null) {
                    JsonNode root = mapper.readTree(data);
                    int choice = root.get("selection").asInt();
                    StringBuilder sb = new StringBuilder();
                    String res = "";
                    long start;
                    long end;

                    switch (choice) {
                        case 0:

sb.append(bc.getChainSize()).append("#").append(bc.getLatestBlock().getDifficulty()).append("#")
                                    .append(bc.getTotalDifficulty()).append("#").append(bc.getHashesPerSeco
nd()).append("#")
                                    .append(bc.getTotalExpectedHashes()).append("#").append(bc.getLatestBlo
ck().getNonce())
                                    .append("#").append(bc.getChainHash());
                            res = rm.toJsonFormat(0, sb.toString());
                            System.out.println("Response: " + res);
                            System.out.println();
                            break;
```

```
                        case 1:
                            System.out.println("Adding a block");
                            Block b = new Block(bc.getChainSize(), bc.getTime(),
root.get("Tx").textValue(), root.get("difficulty").asInt());
                            start = System.currentTimeMillis();
                            bc.addBlock(b);
                            end = System.currentTimeMillis();
                            long time = end - start;
                            sb.append("Total execution time to add this block was ").append(time).append("
milliseconds");
                            System.out.println("Setting response to " + sb);
                            res = rm.toJsonFormat(1, sb.toString());
                            System.out.println("... " + res);
                            System.out.println();
                            break;

                        case 2:
                            System.out.println("Verifying entire chain");
                            start = System.currentTimeMillis();
                            String valid = bc.isChainValid();
                            end = System.currentTimeMillis();
                            sb.append("Total execution time to verify this block was ").append(end -
start).append(" milliseconds");
                            System.out.println("Chain verification: " + valid);
                            System.out.println(sb);
                            System.out.println("Setting response to " + sb);
                            sb.append("#").append(valid);
                            res = rm.toJsonFormat(2, sb.toString());
                            //System.out.println("Response: " + res);
                            System.out.println();
                            break;
                        case 3:
                            System.out.println("View the Blockchain");
                            System.out.println("Setting response to " + bc);
                            res = bc.toString();
                            System.out.println();
                            break;
                        case 4:
                            System.out.println("Corrupt the Blockchain");
```

```
bc.getBlock(root.get("index").asInt()).setData(root.get("newData").textValue());
                            sb.append("Block ").append(root.get("index").asInt()).append(" now holds
").append(root.get("newData").textValue());
                            System.out.println(sb);
                            System.out.println("Setting response to " + sb);
                            res = rm.toJsonFormat(4, sb.toString());
                            System.out.println();
                            break;
                        case 5:
                            System.out.println("Repairing the entire chain");
                            start = System.currentTimeMillis();
                            bc.repairChain();
                            end = System.currentTimeMillis();
                            sb.append("Total execution time to repair this block was ").append(end -
start).append(" milliseconds");
                            System.out.println("Setting response to " + sb);
                            res = rm.toJsonFormat(5, sb.toString());
                            System.out.println();
                            break;

                    }

                    out.println(res);
                    out.flush();
                }
            }

            // Handle exceptions
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());

            // If quitting (typically by you sending quit signal) clean up sockets
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
```

```
    }


}
```

Task 1 Server Source Code (ResponseMessage.java)

```
//Name: Yu Chen
//Andrew Id: yuc3

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;

/**
 * This is the Project 3 Task 1 for 95-702.
 *
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class ResponseMessage {
    /**
     * I use Jackson as the Java JSON library
     * reference 1 : https://www.baeldung.com/jackson-object-mapper-tutorial
     * reference 2 : http://tutorials.jenkov.com/java-json/jackson-objectmapper.html
     *
     * @param selection
     * @param data
     * @return json format response message to send back to client
     * @throws JsonProcessingException
     */
    public String toJsonFormat(int selection, String data) throws JsonProcessingException {
        ObjectMapper mapper = new ObjectMapper();

        ObjectNode outerObject = mapper.createObjectNode();
        outerObject.putPOJO("selection", selection);
        String[] strs = data.split("#");

        switch (selection) {
```

```
            case 0:
                String chainHash = mapper.writeValueAsString(strs[6]);
                outerObject.putPOJO("size", strs[0]);
                outerObject.putPOJO("chainHash", chainHash);
                outerObject.putPOJO("totalHashes", strs[4]);
                outerObject.putPOJO("totalDiff", strs[2]);
                outerObject.putPOJO("recentNonce", strs[5]);
                outerObject.putPOJO("diff", strs[1]);
                outerObject.putPOJO("hps", strs[3]);
                break;
            case 1: //for case 1,4,5 the json field will all be selection and response
            case 4:
            case 5:
                String tx = mapper.writeValueAsString(strs[0]);
                outerObject.putPOJO("response", tx);
                break;
            case 2:
                String time = mapper.writeValueAsString(strs[0]);
                String valid = mapper.writeValueAsString(strs[1]);
                outerObject.putPOJO("valid", valid);
                outerObject.putPOJO("response", time);
                break;
            case 3: //since the BlockChain class's toString method has already format to json string for
us,
                // we could simply do nothing here
                break;

        }

        return outerObject.toString();
    }
}
```

Task 1 Server Source Code (Block.java)

```
//Name: Yu Chen
//Andrew Id: yuc3
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
```

```java
import java.sql.Timestamp;

/**
 * This is the Project 3 for 95-702.
 * The reference doc is: https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/Block.html
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class Block {
    /**
     * Instance variable - the position of the block on the chain.
     */
    private int index;
    /**
     * Instance variable - holds the time of the block's creation.
     */
    private Timestamp timestamp;
    /**
     * Instance variable - holding the block's single transaction details.
     */
    private String data;
    /**
     * Instance variable - the minimum number of left most hex digits needed by a proper hash.
     */
    private int difficulty;
    /**
     * Instance variable - determined by a proof of work routine.
     */
    private BigInteger nonce;
    /**
     * Instance variable - the SHA256 hash of a block's parent(hash pointer).
     */
    private String previousHash;

    /**
     * This the Block constructor.
     * @param index - This is the position within the chain. Genesis is at 0.
     * @param timestamp - This is the time this block was added.
     * @param data - This is the transaction to be included on the blockchain.
     * @param difficulty - This is the number of leftmost nibbles that need to be 0.
     */
```

```
    public Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.difficulty = difficulty;
    }

    /**
     * This method returns the nonce for this block.
     * The nonce is a number that has been found to cause
     * the hash of this block to have the correct number of leading hexadecimal zeroes.
     *
     * @return a BigInteger representing the nonce for this block
     */
    public BigInteger getNonce() {
        return nonce;
    }

    /**
     * Simple getter method.
     *
     * @return difficulty
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * @return data
     */
    public String getData() {
        return data;
    }

    /**
     * @return index
     */
    public int getIndex() {
        return index;
    }
```

```java
    /**
     * @return previous hash
     */
    public String getPreviousHash() {
        return previousHash;
    }

    /**
     * @return timestamp
     */
    public Timestamp getTimestamp() {
        return timestamp;
    }

    /**
     * Simple setter method
     * @param difficulty - determines how much work is required to produce a proper hash
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Simple setter method
     * @param previousHash - a hashpointer to this block's parent
     */
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    /**
     * Simple setter method
     * @param index - the index of this block in the chain
     */
    public void setIndex(int index) {
        this.index = index;

    }

    /**
     * Simple setter method
```

```
     * @param data - represents the transaction held by this block
     */
    public void setData(String data) {
        this.data = data;

    }

    /**
     * Simple setter method
     * @param timestamp - of when this block was created
     */
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }

    /**
     * This method computes a hash of the concatenation of the index, timestamp, data, previousHash, nonce,
and difficulty.
     *
     * @return a String holding Hexadecimal characters
     */
    public String calculateHash() throws NoSuchAlgorithmException {
        StringBuilder sb = new StringBuilder();

sb.append(index).append(timestamp).append(data).append(previousHash).append(nonce).append(difficulty);
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(sb.toString().getBytes());
        return bytesToHex(md.digest());
    }

    // Reference from: 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();

    public 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);
    }

    /**
     * The proof of work methods finds 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.
     */
    public String proofOfWork() throws NoSuchAlgorithmException {
        BigInteger n = new BigInteger("0");
        String leadingZeros = "0".repeat(Math.max(0, difficulty));
        String s = calculateHash();

        while (!s.substring(0, difficulty).equals(leadingZeros)) {
            n = n.add(new BigInteger("1"));
            nonce = n; //set the nonce
            s = calculateHash();
        }

        return calculateHash();
    }


    /**
     * @return A JSON representation of all of this block's data is returned.
     */
    @Override
    public String toString() { //output to a json format
        return "{\"index\": "+ getIndex() +", \"time stamp\": \""+ getTimestamp() +
                "\", \"Tx\": \""+ getData() +"\", \"PrevHash\": \""+ getPreviousHash() +
                "\", \"nonce\": "+ getNonce() + ", \"difficulty\": "+ getDifficulty() +
                "}";
```

```
    }

    public static void main(String[] args){

    }
}
```

Task 1 Server Source Code (BlockChain.java)

```
//Name: Yu Chen
//Andrew Id: yuc3

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

/**
 * This is the Project 3 for 95-702.
 * The reference doc is: https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html
 *
 * @author Jennifer Chen (yuc3@andrew.cmu.edu)
 */
public class BlockChain {
    /**
     * Instance member - an ArrayList to hold Blocks.
     */
    private List<Block> blocks;
    /**
     * Instance variable - a chain hash to hold a SHA256 hash of the most recently added Block.
     */
    private String chainHash;
    /**
     * Instance variable - approximate number of hashes per second on this computer.
     */
    private long hashesPerSecond;
```

```java
    /**
     * Constructor - This BlockChain has exactly three instance members:
     * an ArrayList to hold Blocks and a chain hash to hold a SHA256 hash of the most recently added Block.
     */
    public BlockChain() {
        blocks = new ArrayList<>();
        chainHash = "";
        hashesPerSecond = 0;
    }

    /**
     * A new Block is being added to the BlockChain.
     * This new block's previous hash must hold the hash of the most recently added block.
     * After this call on addBlock, the new block becomes the most recently added block on the BlockChain.
     *
     * @param newBlock - is added to the BlockChain as the most recent block
     * @throws NoSuchAlgorithmException
     */
    public void addBlock(Block newBlock) throws NoSuchAlgorithmException {
        newBlock.setPreviousHash(chainHash);
        newBlock.proofOfWork();
        blocks.add(newBlock);
        chainHash = newBlock.calculateHash();
    }

    /**
     * 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() throws NoSuchAlgorithmException {
        long start = System.currentTimeMillis();
        int i = 0;
        while (i < 2000000) {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update("00000000".getBytes());
            bytesToHex(md.digest());
            i++;
        }
        long end = System.currentTimeMillis();
```

```
        hashesPerSecond = (long) ((long) 2000000 / ((end - start) / 1000.0));
    }

    // Reference from : 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();

    public 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);
    }

    /**
     * @param i
     * @return block at postion i
     */
    public Block getBlock(int i) {
        return blocks.get(i);
    }

    public String getChainHash() {
        return chainHash;
    }

    /**
     * @return the size of the chain in blocks.
     */
    public int getChainSize() {
        return blocks.size();
    }

    /**
     * @return the instance variable approximating the number of hashes per second.
     */
    public int getHashesPerSecond() {
        return (int) hashesPerSecond;
```

```
    }

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

    /**
     * @return the current system time.
     */
    public Timestamp getTime() {
        return new Timestamp(System.currentTimeMillis());
    }

    /**
     * Compute and return the total difficulty of all blocks on the chain. Each block knows its own
difficulty.
     *
     * @return total difficulty
     */
    public int getTotalDifficulty() {
        int res = 0;
        for (Block b : blocks) {
            res += b.getDifficulty();
        }
        return res;
    }

    /**
     * Compute and return the expected number of hashes required for the entire chain.
     *
     * @return getTotalExpectedHashes
     */
    public double getTotalExpectedHashes() {
        double res = 0;
        for (Block b : blocks) {
            int blockPerHash = 1;
            int times = b.getDifficulty();
            for (int i = 0; i < times; i++) {
```

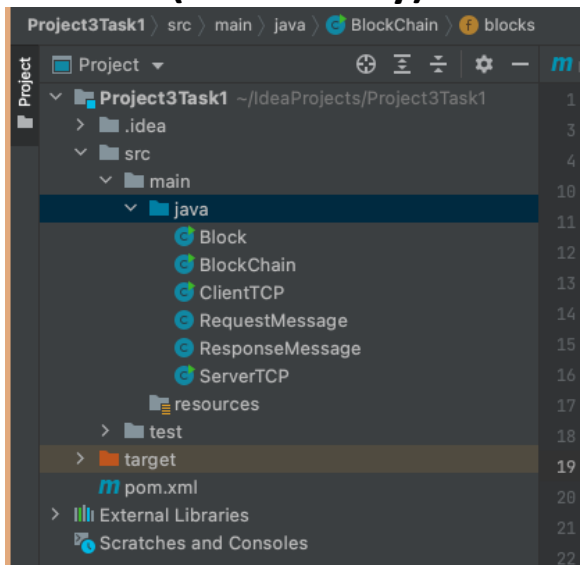
```
                blockPerHash *= 16;
            }
            res += blockPerHash;
        }
        return res;
    }

    /**
     * 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".
     * <p>
     * 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
     * @throws NoSuchAlgorithmException
     */
    public String isChainValid() throws NoSuchAlgorithmException {
        if (blocks.size() == 1) { //if the chain only contains one block
            String hash = blocks.get(0).calculateHash();
            int hashZeros = 0;
            int index = 0;
            while (hash.charAt(index) == '0') {
                hashZeros++;
                index++;
            }
            if (!hash.equals(chainHash) && hashZeros != blocks.get(0).getDifficulty()) {
                return "FALSE! Improper hash on node 0. Does not begin with " +
"0".repeat(blocks.get(0).getDifficulty());
            }
        } else { //if the chain contains more then one block
            for (int i = 0; i < blocks.size(); i++) {
                String hash = blocks.get(i).calculateHash();
                int hashZeros = 0;
```

```
                int index = 0;
                while (hash.charAt(index) == '0') {
                    hashZeros++;
                    index++;
                }
                if (i != blocks.size() - 1 &&
                        !hash.equals(blocks.get(i + 1).getPreviousHash()) && hashZeros !=
blocks.get(i).getDifficulty()) {
                    return "FALSE! Improper hash on node " + i + ". " +
                            "Does not begin with " + "0".repeat(blocks.get(i).getDifficulty());
                }
                if (i == blocks.size() - 1 && // for the last block of the chain
                        !hash.equals(chainHash) && hashZeros != blocks.get(i).getDifficulty()) {
                    return "FALSE! Improper hash on node " + i + ". " +
                            "Does not begin with " + "0".repeat(blocks.get(i).getDifficulty());
                }
            }
        }
        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.
     *
     * @throws NoSuchAlgorithmException
     */
    public void repairChain() throws NoSuchAlgorithmException {
        if (blocks.size() == 1) { //if the chain only contains one block
            String hash = blocks.get(0).calculateHash();
            int hashZeros = 0;
            int index = 0;
            while (hash.charAt(index) == '0') {
                hashZeros++;
                index++;
            }
            if (!hash.equals(chainHash) && hashZeros != blocks.get(0).getDifficulty()) {
                blocks.get(0).proofOfWork(); //need to compute proofOfWork again
```

```
            }
        } else { //if the chain contains more then one block
            for (int i = 0; i < blocks.size(); i++) {
                String hash = blocks.get(i).calculateHash();
                int hashZeros = 0;
                int index = 0;
                while (hash.charAt(index) == '0') {
                    hashZeros++;
                    index++;
                }
                if (i != blocks.size() - 1 &&
                        !hash.equals(blocks.get(i + 1).getPreviousHash()) && hashZeros !=
blocks.get(i).getDifficulty()) {
                    String newHash = blocks.get(i).proofOfWork();
                    blocks.get(i + 1).setPreviousHash(newHash); //reset the hashes
                    blocks.get(i + 1).setTimestamp(getTime()); //reset the time stamp
                }
                if (i == blocks.size() - 1 &&
                        !hash.equals(chainHash) && hashZeros != blocks.get(i).getDifficulty()) {
                    blocks.get(i).proofOfWork();
                    chainHash = blocks.get(i).calculateHash();
                }
            }
        }


    }

    @Override
    public String toString() { //out put to a json format
        StringBuilder sb = new StringBuilder();
        sb.append("{\"ds_chain\" : [ ");
        for (int i = 0; i < blocks.size(); i++) {
            if (i != blocks.size() - 1) {
                sb.append(blocks.get(i)).append(",");
            } else {
                sb.append(blocks.get(i));
            }
        }
        sb.append(" ], \"chainHash\":\"").append(chainHash).append("\"}");
        return sb.toString();
```

```
    }

    public static void main(String[] args){

    }


}
```

Project Summary for Task1: use Maven project because need to add dependency for Jackson (Json library)




```
<dependencies>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.8</version>
    </dependency>
</dependencies>
```