Project 3 Task 0

Task 0 Execution

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55759:/Applications/IntelliJ
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/wantienchiang/IdeaProjects/DS/Project3Task0/target/classes:/Users/wantienchiang/.m2/reposit
ory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar blockchaintask0.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
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: 1092000
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 529
Chain hash: 00C22860A307A1BC017464C29D945D75B00974CAD54359CA1A4B26B9F032388F
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 Bill 100 DSCoin
Total execution time to add this block was 2 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
Bill pays Clara 50 DSCoin
Total execution time to add this block was 2 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
Clara pays Daisy 10 DS Coin
Total execution time to add this block was 2 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":"2023-03-17
19:09:09.858","Tx":"Genesis","PrevHash":"","nonce":529,"difficulty":2},{"index":1,"timestamp":"202
3-03-17 19:10:44.693","Tx":"Alice pays Bill 100
DSCoin","PrevHash":"00C22860A307A1BC017464C29D945D75B00974CAD54359CA1A4B26B9F032388F","nonce":122,
"difficulty":2},{"index":2,"timestamp":"2023-03-17 19:11:36.777","Tx":"Bill pays Clara 50
DSCoin","PrevHash":"0088E5529236CBB4368464B56A3E2F8A4B6B6D75D88B8E644401A17CAF275596","nonce":277,
"difficulty":2},{"index":3,"timestamp":"2023-03-17 19:12:09.203","Tx":"Clara pays Daisy 10 DS
Coin","PrevHash":"00950CCDFDF9344FB1674C47D3D71CCA7C26E06EB3389EDFECB3161B4F0B9ABA","nonce":121,"d
ifficulty":2}],"chainHash":"008D717AAE6814C2B1EBCC011AAAA316E33FD9A2A16DAE64C5B562D5FD970356"}
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 Bill 76 DSCoin
Block 1 now holds Alice pays Bill 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":"2023-03-17
19:09:09.858","Tx":"Genesis","PrevHash":"","nonce":529,"difficulty":2},{"index":1,"timestamp":"202
3-03-17 19:10:44.693","Tx":"Alice pays Bill 76
DSCoin","PrevHash":"00C22860A307A1BC017464C29D945D75B00974CAD54359CA1A4B26B9F032388F","nonce":122,
"difficulty":2},{"index":2,"timestamp":"2023-03-17 19:11:36.777","Tx":"Bill pays Clara 50
DSCoin","PrevHash":"0088E5529236CBB4368464B56A3E2F8A4B6B6D75D88B8E644401A17CAF275596","nonce":277,
"difficulty":2},{"index":3,"timestamp":"2023-03-17 19:12:09.203","Tx":"Clara pays Daisy 10 DS
Coin","PrevHash":"00950CCDFDF9344FB1674C47D3D71CCA7C26E06EB3389EDFECB3161B4F0B9ABA","nonce":121,"d
ifficulty":2}],"chainHash":"008D717AAE6814C2B1EBCC011AAAA316E33FD9A2A16DAE64C5B562D5FD970356"}
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 1Does not begin with 00
Total execution time to verify the chain was 1 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 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 1 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
Daisy pays Sean 25 DSCoin
Total execution time to add this block was 18 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: 1092000
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 5877
Chain hash: 0000291D11762C7EDDF551809438846BD3CAC9E907D19AE52CE677699145B6A2
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

```
/**
 * This class represents a simple Block.
 * @author Candice Chiang
 * Andrew id: wantienc
 * Last Modified: Mar 17, 2023
 */

package blockchaintask0;

// Import the necessary packages
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.Expose;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

public class Block {
    // the index of this block in the chain
    @Expose private int index;
    // of when this block was created
    @Expose private Timestamp timestamp;
    // the transaction
    @Expose private String Tx;
    // the SHA256 hash of a block's parent
    @Expose private String PrevHash;
    // a BigInteger value determined by a proof of work routine
    @Expose private BigInteger nonce;
    // an int that specifies the minimum number of left most hex digits needed by a proper hash
    @Expose private int difficulty;
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();

    /**
     * Constructor.
     * @param index the index of this block in the chain
     * @param timestamp of when this block was created
     * @param data represents the transaction held by this block
     * @param difficulty determines how much work is required to produce a proper hash
     */
    Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.Tx = data;
        this.difficulty = difficulty;
        this.nonce = BigInteger.ZERO;
    }

    /**
     * Computes a hash of the concatenation of the index, timestamp, data, previousHash, nonce,
and difficulty.
     * @return a String holding Hexadecimal characters
     */
    public String calculateHash() {
        String details = getIndex() + getTimestamp().toString() + getData() + getPreviousHash() +
getNonce() + getDifficulty();
        String hash = "";
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-256");
            md.update(details.getBytes());
            hash = bytesToHex(md.digest());
        } catch (NoSuchAlgorithmException e) {
            System.out.println("No hash value available" + e);
        }
        return hash;
    }

    /**
     * Returns a hex string given an array of bytes.
     * Refer to https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-hex-
string-in-java.
     * @param bytes array of bytes to converted
     * @return a hex string
     */
    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);
    }

    /**
     * Get this block's transaction.
     * @return this block's transaction
     */
    public String getData() {
        return this.Tx;
    }

    /**
     * Get difficulty.
     * @return difficulty
     */
    public int getDifficulty() {
        return this.difficulty;
    }

    /**
     * Get index of block.
     * @return index of block
     */
    public int getIndex() {
        return this.index;
    }

    /**
     * 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 this.nonce;
```

```
}

/**
 * Get previous hash.
 * @return previous hash
 */
public String getPreviousHash() {
    return this.PrevHash;
}

/**
 * Get timestamp of this block.
 * @return timestamp of this block
 */
public Timestamp getTimestamp() {
    return this.timestamp;
}

/**
 * Finds a good hash. It increments the nonce until it produces a good hash.
 * @return a String with a hash that has the appropriate number of leading hex zeroes.
 */
public String proofOfWork() {
    String targetLeadingZeroes = "0".repeat(getDifficulty());
    String hash = "";
    while (true) {
        hash = calculateHash();
        if (!hash.substring(0, getDifficulty()).equals(targetLeadingZeroes)) {
            nonce = nonce.add(BigInteger.ONE);
        } else {
            break;
        }
    }
    return hash;
}

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

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

/**
 * Set index.
 * @param index - the index of this block in the chain
 */
public void setIndex(int index) {
    this.index = index;
}
```

```
    /**
     * Set previous hash.
     * @param previousHash - a hashpointer to this block's parent
     */
    public void setPreviousHash(String previousHash) {
        this.PrevHash = previousHash;
    }

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

    /**
     * Block to String.
     * @return A JSON representation of all of this block's data is returned.
     */
    @Override
    public String toString() {
        Gson gson = new GsonBuilder().excludeFieldsWithoutExposeAnnotation().setDateFormat("yyyy-
MM-dd HH:mm:ss.SSS").create();
        return gson.toJson(this);
    }
}
```

Task 0 BlockChain.java

```
/**
 * This class represents a simple BlockChain.
 * @author Candice Chiang
 * Andrew id: wantienc
 * Last Modified: Mar 17, 2023
 */
package blockchaintask0;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.Expose;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Scanner;

public class BlockChain {
    // an ArrayList to hold Blocks
    @Expose private ArrayList<Block> ds_chain;
    // a String to hold a SHA256 hash of the most recently added Block
    @Expose private String chainHash;
    // the instance variable approximating the number of hashes per second
    private int hashPerSecond;
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();
    // menu
    private static final String[] menu = {"View basic blockchain status.",
                                          "Add a transaction to the blockchain.",
                                          "Verify the blockchain.",
                                          "View the blockchain.",
                                          "Corrupt the chain.",
                                          "Hide the corruption by repairing the chain.",
                                          "Exit"};

    /**
     * Constructor.
     */
    BlockChain() {
        this.ds_chain = new ArrayList<>();
        this.chainHash = "";
        this.hashPerSecond = 0;
    }

    /*
    Run times for addBlock increase as the difficulty level gets higher.
    Difficulty <= 3 typically takes less than 10 milliseconds to add a block,
    while >= 4 can take more than 100 milliseconds.
    For isChainValid, it generally takes less than 1 millisecond.
    For chainRepair, similarly, as the difficulty level gets higher,
    it will take longer, as addBlock function.
     */
    public static void main(String[] args) {
        // Initiate the BlockChain and add the first Genesis block with the difficulty of 2
        BlockChain bc = new BlockChain();
        bc.addBlock(new Block(bc.getChainSize(), bc.getTime(), "Genesis", 2));
        bc.computeHashesPerSecond();
        int choice = -1;
        Timestamp start;
        Timestamp end;
        int executionTime;
        Scanner scn = new Scanner(System.in);
        // Continue to get user selection until user selects to quit
        while (choice != 6) {
            // Display menu
            for (int i = 0; i < menu.length; i++) {
                System.out.print(i + ". ");
                System.out.println(menu[i]);
            }
            // Get user choice
            choice = scn.nextInt();
            switch (choice) {
                case 0 -> { // If user selects to view the blockchain status
                    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());
                }
                case 1 -> { // If user selects to add a block
                    System.out.println("Enter difficulty > 0");
                    int difficulty = scn.nextInt();
                    System.out.println("Enter transaction");
                    scn.nextLine();
                    String data = scn.nextLine();
                    start = bc.getTime();
                    bc.addBlock(new Block(bc.getChainSize(), bc.getTime(), data, difficulty));
                    end = bc.getTime();
                    executionTime = (int) (end.getTime() - start.getTime());
                    System.out.println("Total execution time to add this block was " +
executionTime + " milliseconds");
                }
                case 2 -> { // If user selects to verify the blockchain
                    start = bc.getTime();
                    String result = bc.isChainValid();
                    end = bc.getTime();
                    executionTime = (int) (end.getTime() - start.getTime());
                    System.out.print("Chain verification: ");
                    // Print verification result
                    if (result.equals("TRUE")) {
                        System.out.println(result);
                    } else { // False with additional error message
                        System.out.println("FALSE");
                        System.out.println(result);
                    }
                    System.out.println("Total execution time to verify the chain was " +
executionTime + " milliseconds");
                }
                case 3 -> { // If user selects to view the blockchain
                    System.out.println("View the Blockchain");
                    System.out.println(bc);
                }
                case 4 -> { // If user selects to corrupt the blockchain
                    System.out.println("corrupt the Blockchain");
                    System.out.println("Enter block ID of block to corrupt");
                    int id = scn.nextInt();
                    System.out.println("Enter new data for block " + id);
                    scn.nextLine();
                    String newData = scn.nextLine();
                    bc.getBlock(id).setData(newData);
                    System.out.println("Block " + id + " now holds " + newData);
                }
                case 5 -> { // If user selects to repair the blockchain
                    start = bc.getTime();
                    bc.repairChain();
                    end = bc.getTime();
                    executionTime = (int) (end.getTime() - start.getTime());
                    System.out.println("Total execution time required to repair the chain was " +
executionTime + " milliseconds");
                }
                default -> {
                }
            }
        }
        scn.close();
    }

    /**
     * Add a new Block to the BlockChain.
     * @param newBlock - is added to the BlockChain as the most recent block
     */
    public void addBlock(Block newBlock) {
        newBlock.setPreviousHash(this.chainHash);
        this.chainHash = newBlock.proofOfWork();
        this.ds_chain.add(newBlock);
    }

    /**
     * Computes exactly 2 million hashes and times how long that process takes.
     */
    public void computeHashesPerSecond() {
        String str = "00000000";
        Timestamp start = getTime();
        for (int i = 0; i < 2000000; i++) {
            calculateHash(str);
        }
        Timestamp end = getTime();
        this.hashPerSecond = (int) ((2000000 / (end.getTime() - start.getTime())) * 1000);
    }

    /**
     * Return block at position i.
     * @param i position
     * @return Block at position i
     */
    public Block getBlock(int i) {
        return this.ds_chain.get(i);
    }

    /**
     * Get chain hash.
     * @return chain hash
     */
    public String getChainHash() {
        return this.chainHash;
    }

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

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

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

    /**
     * Get the current system time.
     * @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 totalDifficulty
     */
    public int getTotalDifficulty() {
        int totalDifficulty = 0;
        for (Block block: ds_chain) {
            totalDifficulty += block.getDifficulty();
        }
        return totalDifficulty;
    }

    /**
     * Compute and return the expected number of hashes required for the entire chain.
     * @return totalExpectedHashes
     */
    public double getTotalExpectedHashes() {
        double totalExpectedHashes = 0;
        for (Block block: ds_chain) {
            totalExpectedHashes += Math.pow(16, block.getDifficulty());
        }
        return totalExpectedHashes;
    }

    /**
     * Verify if the BlockChain is valid.
     * A valid BlockChain should satisfy:
     * 1. the hash of each block has the requisite number of leftmost 0's (proof of work) as
specified in the difficulty field.
     * 2. the chain hash is equal to this computed hash.
     * @return "TRUE" if the chain is valid, otherwise an error message
     */
    public String isChainValid() {
        String previousHash = "";
        for (int i = 0; i < getChainSize(); i++) {
            Block b = getBlock(i);
            String hash = b.calculateHash();
            int result = isBlockValid(b, hash, previousHash);
            if (result == -1) {
                return "Improper hash on node " + i + "Does not begin with " +
"0".repeat(b.getDifficulty());
            } else if (result == -2) {
                return "Chain hash is not correct";
            }
            previousHash = hash;
        }
        return "TRUE";
    }

    /**
     * Repairs the chain.
     * It checks the hashes of each block and ensures that any illegal hashes are recomputed.
     * Also, it computes new proof of work based on the difficulty specified in the Block.
     */
    public void repairChain() {
        String previousHash = "";
        for (int i = 0;i < getChainSize(); i++) {
            Block b = getBlock(i);
            String hash = b.calculateHash();
            if (isBlockValid(b, hash, previousHash) != 0) {
                if (i < getChainSize() - 1) {
                    getBlock(i + 1).setPreviousHash(b.proofOfWork());
                } else {
                    this.chainHash = b.proofOfWork();
                }
            }
            previousHash = hash;
        }
    }

    /**
     * Helper method of isChainValid and repairChain.
     * Verify if a Block is valid.
     * @param block Block to verify
     * @param hash hash value
     * @param previousHash previous hash
     * @return 0 if valid, -1 if not beginning with the requisite number, -2 if chain hash is
incorrect
     */
    public int isBlockValid(Block block, String hash, String previousHash) {
        String proof = "0".repeat(block.getDifficulty());
        if (!hash.substring(0, block.getDifficulty()).equals(proof)) {
            return -1;
        }
        if (!block.getPreviousHash().equals(previousHash)) {
            return -2;
        }
        return 0;
    }

    /**
     * Uses the toString method defined on each individual block.
     * @return a String representation of the entire chain is returned
     */
    @Override
    public String toString() {
        Gson gson = new GsonBuilder().excludeFieldsWithoutExposeAnnotation().setDateFormat("yyyy-
MM-dd HH:mm:ss.SSS").create();
        return gson.toJson(this);
    }

    /**
     * Computes a hash of the concatenation of the index, timestamp, data, previousHash, nonce,
and difficulty.
     * @param toHash String to hash
     * @return a String holding Hexadecimal characters
     */
    public String calculateHash(String toHash) {
        String hash = "";
        MessageDigest md;
        try {
            md = MessageDigest.getInstance("SHA-256");
            md.update(toHash.getBytes());
            hash = bytesToHex(md.digest());
        } catch (NoSuchAlgorithmException e) {
            System.out.println("No hash value available" + e);
        }
        return hash;
    }
    /**
     * Returns a hex string given an array of bytes.
     * Refer to https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-hex-
string-in-java.
     * @param bytes array of bytes to converted
     * @return a hex string
     */
    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);
    }
}
```

Project 3 Task 1

Task 1 Client Side Execution

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55909:/Applications/IntelliJ
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/wantienchiang/IdeaProjects/DS/Project3Task1/target/classes:/Users/wantienchiang/.m2/reposit
ory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar blockchaintask1.RequestMessage
Blockchain client running.
Please enter 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: 1308000
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 689
Chain hash: 00A015D38FA733C81C76F48897FB9B6838B49B6F1DEBCF3F8E13A4E2AA38B255
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 Bill 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
Bill pays Clara 50 DSCoin
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
1
Enter difficulty > 0
2
Enter transaction
Clara pays Daisy 10 DS Coin
Total execution time to add this block was 2 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":"2023-03-17
19:22:58.351","Tx":"Genesis","PrevHash":"","nonce":689,"difficulty":2},{"index":1,"timestamp":"202
3-03-17 19:23:30.358","Tx":"Alice pays Bill 100
DSCoin","PrevHash":"00A015D38FA733C81C76F48897FB9B6838B49B6F1DEBCF3F8E13A4E2AA38B255","nonce":320,
"difficulty":2},{"index":2,"timestamp":"2023-03-17 19:23:43.978","Tx":"Bill pays Clara 50
DSCoin","PrevHash":"00FDB234511456968B0D6A20E33C9EF93894A4289E7B285B43B78C19C1AF4AF1","nonce":27,"
difficulty":2},{"index":3,"timestamp":"2023-03-17 19:23:58.008","Tx":"Clara pays Daisy 10 DS
Coin","PrevHash":"00DF14A57843DC5457D842E0A59FB707FFF3B899BCD8A51FBCA6D44B30484ED4","nonce":220,"d
ifficulty":2}],"chainHash":"001316805788E84DF8B5E64185FBDFE7769903A644BFAFBB48C126B8C5777E17"}
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 Bill 76 DSCoin
Block 1 now holds Alice pays Bill 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":"2023-03-17
19:22:58.351","Tx":"Genesis","PrevHash":"","nonce":689,"difficulty":2},{"index":1,"timestamp":"202
3-03-17 19:23:30.358","Tx":"Alice pays Bill 76
DSCoin","PrevHash":"00A015D38FA733C81C76F48897FB9B6838B49B6F1DEBCF3F8E13A4E2AA38B255","nonce":320,
"difficulty":2},{"index":2,"timestamp":"2023-03-17 19:23:43.978","Tx":"Bill pays Clara 50
DSCoin","PrevHash":"00FDB234511456968B0D6A20E33C9EF93894A4289E7B285B43B78C19C1AF4AF1","nonce":27,"
difficulty":2},{"index":3,"timestamp":"2023-03-17 19:23:58.008","Tx":"Clara pays Daisy 10 DS
Coin","PrevHash":"00DF14A57843DC5457D842E0A59FB707FFF3B899BCD8A51FBCA6D44B30484ED4","nonce":220,"d
ifficulty":2}],"chainHash":"001316805788E84DF8B5E64185FBDFE7769903A644BFAFBB48C126B8C5777E17"}
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 1Does not begin with 00
Total execution time to verify the chain was 1 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 2 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 1 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
Daisy pays Sean 25 DSCoin
Total execution time to add this block was 351 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: 1308000
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 82716
Chain hash: 00009CC469FDDF49D5BA7DE627A8683303137F137E91D4FE38746D1513CFFB10
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

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=55904:/Applications/IntelliJ
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/wantienchiang/IdeaProjects/DS/Project3Task1/target/classes:/Users/wantienchiang/.m2/reposit
ory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar blockchaintask1.ResponseMessage
Blockchain server running
We have a visitor
Response:
{"selection":0,"size":1,"chainHash":"00A015D38FA733C81C76F48897FB9B6838B49B6F1DEBCF3F8E13A4E2AA38B
255","totalHashes":256.0,"totalDiff":2,"recentNonce":689,"diff":2,"hps":1308000}
Adding a block
Setting response to Total execution time to add this block was 3 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 3 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 0 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 0 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 2 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 2 milliseconds"}
Verifying entire chain
Chain verification: TRUE
Total execution time to verify the chain was 0 milliseconds
Setting response to Total execution time to verify the chain was 0 milliseconds
View the Blockchain
Setting response to {"ds_chain":[{"index":0,"timestamp":"2023-03-17
19:22:58.351","Tx":"Genesis","PrevHash":"","nonce":689,"difficulty":2},{"index":1,"timestamp":"202
3-03-17 19:23:30.358","Tx":"Alice pays Bill 100
DSCoin","PrevHash":"00A015D38FA733C81C76F48897FB9B6838B49B6F1DEBCF3F8E13A4E2AA38B255","nonce":320,
"difficulty":2},{"index":2,"timestamp":"2023-03-17 19:23:43.978","Tx":"Bill pays Clara 50
DSCoin","PrevHash":"00FDB234511456968B0D6A20E33C9EF93894A4289E7B285B43B78C19C1AF4AF1","nonce":27,"
difficulty":2},{"index":3,"timestamp":"2023-03-17 19:23:58.008","Tx":"Clara pays Daisy 10 DS
Coin","PrevHash":"00DF14A57843DC5457D842E0A59FB707FFF3B899BCD8A51FBCA6D44B30484ED4","nonce":220,"d
ifficulty":2}],"chainHash":"001316805788E84DF8B5E64185FBDFE7769903A644BFAFBB48C126B8C5777E17"}
Corrupt the Blockchain
Block 1 now holds Alice pays Bill 76 DSCoin
Setting response to Block 1 now holds Alice pays Bill 76 DSCoin
View the Blockchain
Setting response to {"ds_chain":[{"index":0,"timestamp":"2023-03-17
19:22:58.351","Tx":"Genesis","PrevHash":"","nonce":689,"difficulty":2},{"index":1,"timestamp":"202
3-03-17 19:23:30.358","Tx":"Alice pays Bill 76
DSCoin","PrevHash":"00A015D38FA733C81C76F48897FB9B6838B49B6F1DEBCF3F8E13A4E2AA38B255","nonce":320,
"difficulty":2},{"index":2,"timestamp":"2023-03-17 19:23:43.978","Tx":"Bill pays Clara 50
DSCoin","PrevHash":"00FDB234511456968B0D6A20E33C9EF93894A4289E7B285B43B78C19C1AF4AF1","nonce":27,"
difficulty":2},{"index":3,"timestamp":"2023-03-17 19:23:58.008","Tx":"Clara pays Daisy 10 DS
Coin","PrevHash":"00DF14A57843DC5457D842E0A59FB707FFF3B899BCD8A51FBCA6D44B30484ED4","nonce":220,"d
ifficulty":2}],"chainHash":"001316805788E84DF8B5E64185FBDFE7769903A644BFAFBB48C126B8C5777E17"}
Verifying entire chain
Chain verification: FALSE
Improper hash on node 1Does not begin with 00
Total execution time to verify the chain was 1 milliseconds
Setting response to Total execution time to verify the chain was 1 milliseconds
Repairing the entire chain
Setting response to Total execution time required to repair the chain was 2 milliseconds
Verifying entire chain
Chain verification: TRUE
Total execution time to verify the chain was 1 milliseconds
Setting response to Total execution time to verify the chain was 1 milliseconds
```

```
Adding a block
Setting response to Total execution time to add this block was 351 milliseconds
...{"selection":1,"response":"Total execution time to add this block was 351 milliseconds"}
Response:
{"selection":0,"size":5,"chainHash":"00009CC469FDDF49D5BA7DE627A8683303137F137E91D4FE38746D1513CFF
B10","totalHashes":66560.0,"totalDiff":12,"recentNonce":82716,"diff":4,"hps":1308000}
```

Task 1 Client Source Code

```java
/**
 * This program implements a TCP client.
 * @author Candice Chiang
 * Andrew id: wantienc
 * Last Modified: Mar 15, 2023
 */
package blockchaintask1;

import com.google.gson.Gson;
import java.io.*;
import java.net.InetAddress;
import java.net.Socket;
import java.util.Scanner;

public class RequestMessage {
    // Declare a client socket
    static Socket clientSocket = null;
    // Declare a BufferedReader to read from client socket
    static BufferedReader in = null;
    // Declare a PrintWriter to write to client socket
    static PrintWriter out = null;
    // Destination server port number
    static int serverPort;
    // Host name
    static InetAddress aHost;
    // menu
    private static final String[] menu = {"View basic blockchain status.",
            "Add a transaction to the blockchain.",
            "Verify the blockchain.",
            "View the blockchain.",
            "Corrupt the chain.",
            "Hide the corruption by repairing the chain.",
            "Exit"};
    // user selection
    int selection;
    // block index
    int index;
    // transaction
    String data;
    // // an int that specifies the minimum number of left most hex digits needed by a proper hash
    int difficulty;

    /**
     * Constructor.
     */
    RequestMessage() {
        this.selection = -1;
        this.data = "";
        this.difficulty = 0;
    }
    /**
     * Implement a TCP client.
     * @param args Array of strings giving message contents and server hostname
     */
    public static void main(String[] args) {
        // Announce the client starts running
        System.out.println("Blockchain client running.");
        // Get the server side port number from user
        // For this project, use 6789
        Scanner readInput = new Scanner(System.in);
        System.out.println("Please enter server port: ");
        serverPort = readInput.nextInt();
        Gson gson = new Gson();
        RequestMessage m = new RequestMessage();
        try {
            // Collect the IP address
            aHost = InetAddress.getByName("localhost");
            // Initialize socket
            clientSocket = new Socket(aHost, serverPort);
            Scanner scn = new Scanner(System.in);
            // Continue to get user selection until user selects to quit
            while (m.selection != 6) {
                m = new RequestMessage();
                // Display menu
                for (int i = 0; i < menu.length; i++) {
                    System.out.print(i + ". ");
                    System.out.println(menu[i]);
                }
                // Get user selection
                m.selection = scn.nextInt();
                switch (m.selection) {
                    case 1 -> {  // If user selects to view the blockchain status
                        System.out.println("Enter difficulty > 0");
                        m.difficulty = scn.nextInt();
                        System.out.println("Enter transaction");
                        scn.nextLine();
                        m.data = scn.nextLine();
                    }
                    case 4 -> { // If user selects to add a block
                        System.out.println("corrupt the Blockchain");
                        System.out.println("Enter block ID of block to corrupt");
                        m.index = scn.nextInt();
                        System.out.println("Enter new data for block " + m.index);
                        scn.nextLine();
                        m.data = scn.nextLine();
                    }
                    default -> {
                    }
                }
                // Send request to server except user selects to quit
                if (m.selection != 6) parseRequest(m.selection, gson.toJson(m));
            }
            scn.close();

        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            try {
                // Close socket if not null
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    public static void parseRequest (int selection, String requestStr) {
        // int result to record the reply sum
        ResponseMessage r;
        Gson gson = new Gson();
        try {
            in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
            // Write request to server
            out.println(requestStr);
            out.flush();
            r = gson.fromJson(in.readLine(), ResponseMessage.class); // read a line of data from
the stream
            // Handle IO exceptions
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        switch (selection) {
            case 0 -> { // If user selects to view the blockchain status
                System.out.println("Current size of chain: " + r.size);
                System.out.println("Difficulty of most recent block: " + r.diff);
                System.out.println("Total difficulty for all blocks: " + r.totalDiff);
                System.out.println("Approximate hashes per second on this machine: " + r.hps);
                System.out.println("Expected total hashes required for the whole chain: " +
r.totalHashes);
                System.out.println("Nonce for most recent block: " + r.recentNonce);
                System.out.println("Chain hash: " + r.chainHash);
            }
            // Print the response if user selects to add a block, corrupt the chain, or repair the
chain
            case 1, 4, 5 -> System.out.println(r.response);
            case 2 -> { // // If user selects to verify the blockchain
                System.out.print("Chain verification: ");
                if (!r.errorM.equals("TRUE")) {
                    System.out.println("FALSE");
                }
                System.out.println(r.errorM);
                System.out.println(r.response);
            }
            case 3 -> { // If user selects to view the blockchain
                System.out.println("View the Blockchain");
                System.out.println(r.response);
            }
            default -> {
            }
        }
    }
}
```

Task 1 Server Source Code

```
/**
 * This program implements a TCP server.
 * @author Candice Chiang
 * Andrew id: wantienc
 * Last Modified: Mar 15, 2023
 */
package blockchaintask1;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.Expose;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.math.BigInteger;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Scanner;

public class ResponseMessage {
    // BlockChain object with an ArrayList to hold Blocks and a chain hash to hold a SHA256 hash
of the most recently added Block.
    static BlockChain ds_chain;
    // user selection
    int selection;
    // size of chain
    Integer size;
    // a String to hold a SHA256 hash of the most recently added Block
    String chainHash;
    // the expected number of hashes required for the entire chain
    Double totalHashes;
    // Total difficulty of the blockchain
    Integer totalDiff;
    // a BigInteger value determined by a proof of work routine
    BigInteger recentNonce;
    // an Integer that specifies the minimum number of left most hex digits needed by a proper
hash
    Integer diff;
    // the instance variable approximating the number of hashes per second
    Integer hps;
    // error message if the chain is invalid
    String errorM;
    // blockchain server response
    String response;

    public static void main(String[] args) {
        // Announce the server starts running
        System.out.println("Blockchain server running");
        // Port number this server to listen on
        int serverPort = 6789;
        // Declare client socket
        Socket clientSocket = null;
        ServerSocket listenSocket;
        ds_chain = new BlockChain();
        ds_chain.addBlock(new Block(ds_chain.getChainSize(), ds_chain.getTime(), "Genesis", 2));
        ds_chain.computeHashesPerSecond();
        try {
            // Create a new server socket
            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.
             */
            clientSocket = listenSocket.accept();
            // If we get here, then we are now connected to a client.
            System.out.println("We have a visitor");
            // 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())));
            Gson gson = new Gson();
            RequestMessage m;
            // An infinite loop to wait for incoming requests
            while(true){
                if (in.hasNextLine()) { // if there exists a request
                    // Get request
                    m = gson.fromJson(in.nextLine(), RequestMessage.class);
                    String responseJson = process(m.selection, m);
                    // Write sum result to socket
                    out.println(responseJson);
                    out.flush();

                } else { // Ready to accept another new connection request from client
                    clientSocket = listenSocket.accept();
                    System.out.println("We have a visitor");
                    in = new Scanner(clientSocket.getInputStream());
                    out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
                }

            }

            // 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) { // Close socket if not null
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    /**
     * Process users request and return reponse
     * @param selection user selection
     * @param m request message
     * @return response
     */
    public static String process(int selection, RequestMessage m) {
        Gson gson = new Gson();
        ResponseMessage r = new ResponseMessage();
        Timestamp start;
        Timestamp end;
        int executionTime;
        switch (selection) {
            case 0 -> { // If user selects to view the blockchain status
                r.selection = selection;
                r.size = ds_chain.getChainSize();
                r.chainHash = ds_chain.getChainHash();
                r.totalHashes = ds_chain.getTotalExpectedHashes();
                r.totalDiff = ds_chain.getTotalDifficulty();
                r.recentNonce = ds_chain.getLatestBlock().getNonce();
                r.diff = ds_chain.getLatestBlock().getDifficulty();
                r.hps = ds_chain.hashPerSecond;
                System.out.println("Response: " + gson.toJson(r));
            }
            case 1 -> { // If user selects to add a block
                r.selection = selection;
                System.out.println("Adding a block");
                start = ds_chain.getTime();
                ds_chain.addBlock(new Block(ds_chain.getChainSize(), ds_chain.getTime(), m.data,
m.difficulty));
                end = ds_chain.getTime();
                executionTime = (int) (end.getTime() - start.getTime());
                String t = "Total execution time to add this block was " + executionTime + "
milliseconds";
                System.out.println("Setting response to " + t);
                r.response = t;
                System.out.println("..." + gson.toJson(r));
            }
            case 2 -> { // If user selects to verify the chain
                System.out.println("Verifying entire chain");
                start = ds_chain.getTime();
                String result = ds_chain.isChainValid();
                end = ds_chain.getTime();
                executionTime = (int) (end.getTime() - start.getTime());
                System.out.print("Chain verification: ");
                if (result.equals("TRUE")) {
                    System.out.println("TRUE");
                } else {
                    System.out.println("FALSE");
                    System.out.println(result);
                }
                r.errorM = result;
                String t1 = "Total execution time to verify the chain was " + executionTime + "
milliseconds";
                System.out.println(t1);
                System.out.println("Setting response to " + t1);
                r.response = t1;
            }
            case 3 -> { // If user selects to view the blockchain
                System.out.println("View the Blockchain");
                String view = ds_chain.toString();
                System.out.println("Setting response to " + view);
                r.response = view;
            }
            case 4 -> { // If user selects to corrupt the chain
                System.out.println("Corrupt the Blockchain");
                ds_chain.getBlock(m.index).setData(m.data);
                String newM = "Block " + m.index + " now holds " + m.data;
                System.out.println(newM);
                System.out.println("Setting response to " + newM);
                r.response = newM;
            }
            case 5 -> { // If user selects to repair the chain
                System.out.println("Repairing the entire chain");
                start = ds_chain.getTime();
                ds_chain.repairChain();
                end = ds_chain.getTime();
                executionTime = (int) (end.getTime() - start.getTime());
                String t2 = "Total execution time required to repair the chain was " +
executionTime + " milliseconds";
                System.out.println("Setting response to " + t2);
                r.response = t2;
            }
        }
        return gson.toJson(r);
    }

    public static class BlockChain {
        // an ArrayList to hold Blocks
        @Expose private ArrayList<Block> ds_chain;
        // a String to hold a SHA256 hash of the most recently added Block
        @Expose private String chainHash;
        // the instance variable approximating the number of hashes per second
        private int hashPerSecond;
        private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();
        /**
         * Constructor.
         */
        BlockChain() {
            this.ds_chain = new ArrayList<>();
            this.chainHash = "";
            this.hashPerSecond = 0;
        }
        /**
         * Add a new Block to the BlockChain.
         * @param newBlock - is added to the BlockChain as the most recent block
         */
        public void addBlock(Block newBlock) {
            newBlock.setPreviousHash(this.chainHash);
            this.chainHash = newBlock.proofOfWork();
            this.ds_chain.add(newBlock);
        }
        /**
         * Computes exactly 2 million hashes and times how long that process takes.
         */
        public void computeHashesPerSecond() {
            String str = "00000000";
            Timestamp start = getTime();
            for (int i = 0; i < 2000000; i++) {
                calculateHash(str);
            }
            Timestamp end = getTime();
            this.hashPerSecond = (int) ((2000000 / (end.getTime() - start.getTime())) * 1000);
        }
        /**
         * Return block at position i.
         * @param i position
         * @return Block at position i
         */
        public Block getBlock(int i) {
            return this.ds_chain.get(i);
        }
        /**
         * Get chain hash.
         * @return chain hash
         */
        public String getChainHash() {
            return this.chainHash;
        }
        /**
         * Get the size of the chain in blocks.
         * @return the size of the chain in blocks
         */
        public int getChainSize() {
            return this.ds_chain.size();
        }
        /**
         * Get the instance variable approximating the number of hashes per second.
         * @return the instance variable approximating the number of hashes per second
         */
        public int getHashesPerSecond() {
            return this.hashPerSecond;
        }
        /**
         * Get a reference to the most recently added Block.
         * @return a reference to the most recently added Block
         */
        public Block getLatestBlock() {
            return this.ds_chain.get(this.getChainSize() - 1);
        }
        /**
         * Get the current system time.
         * @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 totalDifficulty
         */
        public int getTotalDifficulty() {
            int totalDifficulty = 0;
            for (Block block: ds_chain) {
                totalDifficulty += block.getDifficulty();
            }
            return totalDifficulty;
        }
        /**
         * Compute and return the expected number of hashes required for the entire chain.
         * @return totalExpectedHashes
         */
        public double getTotalExpectedHashes() {
            double totalExpectedHashes = 0;
            for (Block block: ds_chain) {
                totalExpectedHashes += Math.pow(16, block.getDifficulty());
            }
            return totalExpectedHashes;
        }
        /**
         * Verify if the BlockChain is valid.
         * A valid BlockChain should satisfy:
         * 1. the hash of each block has the requisite number of leftmost 0's (proof of work) as
specified in the difficulty field.
         * 2. the chain hash is equal to this computed hash.
         * @return "TRUE" if the chain is valid, otherwise an error message
         */
        public String isChainValid() {
            String previousHash = "";
            for (int i = 0; i < getChainSize(); i++) {
                Block b = getBlock(i);
                String hash = b.calculateHash();
                int result = isBlockValid(b, hash, previousHash);
                if (result == -1) {
                    return "Improper hash on node " + i + "Does not begin with " +
"0".repeat(b.getDifficulty());
                } else if (result == -2) {
                    return "Chain hash is not correct";
                }
                previousHash = hash;
            }
            return "TRUE";
        }
        /**
         * Repairs the chain.
         * It checks the hashes of each block and ensures that any illegal hashes are recomputed.
         * Also, it computes new proof of work based on the difficulty specified in the Block.
         */
        public void repairChain() {
            String previousHash = "";
            for (int i = 0;i < getChainSize(); i++) {
                Block b = getBlock(i);
                String hash = b.calculateHash();
                if (isBlockValid(b, hash, previousHash) != 0) {
                    if (i < getChainSize() - 1) {
                        getBlock(i + 1).setPreviousHash(b.proofOfWork());
                    } else {
                        this.chainHash = b.proofOfWork();
                    }
                }
                previousHash = hash;
            }
        }
        /**
         * Helper method of isChainValid and repairChain.
         * Verify if a Block is valid.
         * @param block Block to verify
         * @param hash hash value
         * @param previousHash previous hash
         * @return 0 if valid, -1 if not beginning with the requisite number, -2 if chain hash is
incorrect
         */
        public int isBlockValid(Block block, String hash, String previousHash) {
            String proof = "0".repeat(block.getDifficulty());
            if (!hash.substring(0, block.getDifficulty()).equals(proof)) {
                return -1;
            }
            if (!block.getPreviousHash().equals(previousHash)) {
                return -2;
            }
            return 0;
        }
        /**
         * Uses the toString method defined on each individual block.
         * @return a String representation of the entire chain is returned
         */
        @Override
        public String toString() {
            Gson gson = new
GsonBuilder().excludeFieldsWithoutExposeAnnotation().setDateFormat("yyyy-MM-dd
HH:mm:ss.SSS").create();
            return gson.toJson(this);
        }
        /**
         * Computes a hash of the concatenation of the index, timestamp, data, previousHash,
nonce, and difficulty.
         * @param toHash String to hash
         * @return a String holding Hexadecimal characters
         */
        public String calculateHash(String toHash) {
            String hash = "";
            MessageDigest md;
            try {
                md = MessageDigest.getInstance("SHA-256");
                md.update(toHash.getBytes());
                hash =  bytesToHex(md.digest());
            } catch (NoSuchAlgorithmException e) {
                System.out.println("No hash value available" + e);
            }
            return hash;
        }
        /**
         * Returns a hex string given an array of bytes.
         * Refer to https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-
hex-string-in-java.
         * @param bytes array of bytes to converted
         * @return a hex string
         */
        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);
        }
    }
}
```

Project 3 Task 2

```
###
GET
https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/KXQ7ERQWUWCF5I2PEHGBUP24ICYSO3VKDAU
XYUER5W424R5SB3EQ

https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/KXQ7ERQWUWCF5I2PEHGBUP24ICYSO3VKDAU
XYUER5W424R5SB3EQ

HTTP/1.1 200 OK
server: nginx
date: Fri, 17 Mar 2023 23:39:51 GMT
content-type: application/json; charset=UTF-8
content-length: 728
vary: Origin
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-
Stats, Authorization
cache-control: no-store, no-cache, must-revalidate, private
```

```
{
  "current-round": 28481712,
  "transaction": {
    "close-rewards": 0,
    "closing-amount": 0,
    "confirmed-round": 28340980,
    "fee": 1000,
    "first-valid": 28340978,
    "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "genesis-id": "testnet-v1.0",
    "id": "KXQ7ERQWUWCF5I2PEHGBUP24ICYSO3VKDAUXYUER5W424R5SB3EQ",
    "intra-round-offset": 5,
    "last-valid": 28341978,
    "payment-transaction": {
      "amount": 10000000,
      "close-amount": 0,
      "receiver": "B2V2JNLMHRO7ZBSRYD7D3V4LZYR5HZ4NCKVUHT6VEOITLYUOCL75FKYO3U"
    },
    "receiver-rewards": 0,
    "round-time": 1678586064,
    "sender": "DISPE57MNLYKOMOK3H5IMBAYOYW3YL2CSI6MDOG3RDXSMET35DG4W6SOTI",
    "sender-rewards": 0,
    "signature": {
      "sig":
"3oxsKuFZkyHIR4H25jpK3KfUFLi0JnbCLOZ9rS1ek9F15P1c1Uk4SH1uY3G0N+9ub0/AXQw87ONUhNlgOQCUAA=="
    },
    "tx-type": "pay"
  }
}
```

```
Response file saved.
> 2023-03-17T193951.200.json

Response code: 200 (OK); Time: 381ms (381 ms); Content length: 728 bytes (728 B)
```

```
###
GET
https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/XJK2CNHUHF2WAVRVBWGVDPTJ6TZNVBHJTIM
UKTSN6JOE72EZYO6A


https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/XJK2CNHUHF2WAVRVBWGVDPTJ6TZNVBHJTIM
UKTSN6JOE72EZYO6A

HTTP/1.1 200 OK
server: nginx
date: Fri, 17 Mar 2023 23:35:51 GMT
content-type: application/json; charset=UTF-8
content-length: 757
vary: Origin
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-
Stats, Authorization
cache-control: no-store, no-cache, must-revalidate, private
```

```
{
  "current-round": 28481645,
  "transaction": {
    "close-rewards": 0,
    "closing-amount": 0,
    "confirmed-round": 28341030,
    "fee": 1000,
    "first-valid": 28341028,
    "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "genesis-id": "testnet-v1.0",
    "id": "XJK2CNHUHF2WAVRVBWGVDPTJ6TZNVBHJTIMUKTSN6JOE72EZYO6A",
    "intra-round-offset": 1,
    "last-valid": 28342028,
    "note": "UHJvamVjdDNUYXNrMg==",
    "payment-transaction": {
      "amount": 5000000,
      "close-amount": 0,
      "receiver": "K2EP3LIPR3KEI7QOVW3UHLN6JGASMF442YRI5IPO6N6UWPUVNZJ6BVFT4U"
    },
    "receiver-rewards": 0,
    "round-time": 1678586245,
    "sender": "B2V2JNLMHRO7ZBSRYD7D3V4LZYR5HZ4NCKVUHT6VEOITLYUOCL75FKYO3U",
    "sender-rewards": 0,
    "signature": {
      "sig":
"GTQwKQqU79F1qZaX1Huzdb986/RMHNE+TmDqHCZTVHAYUZlEtdZ6wRGhFCKcLXanTdAkpHKaywxdw+DrmMrQCw=="
    },
    "tx-type": "pay"
  }
}
```

```
Response file saved.
> 2023-03-17T193552.200.json

Response code: 200 (OK); Time: 455ms (455 ms); Content length: 757 bytes (757 B)
```