95702: Distributed Systems

Project 3

Name: Ariane Correa
Andrew ID: ajcorrea

Task 0

Console Output:

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52522:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project3Task0/target/classes:/Users/arianecorrea/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.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: 3205
Expected total hashes required for the whole chain: 256.0
Nonce for the most recent block: 216
Chain hash:  0073cf07aeacb687592854d2b6b6015c15f4dd8c7d64771c52e910413e59e0da

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:
Mike pays Marty 100 DSCoin
Total execution time to add this block was 7 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:
Marty pays Joe 50 DSCoin
Total execution time to add this block was 3 milliseconds

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

1
Enter difficulty > 0
2
Enter transaction:
Joe pays Andy 10 DS Coin
Total execution time to add this block was 10 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

3
View the Blockchain
[{"index":0,"timestamp":"2023-10-27
15:19:00.02","tx":"Genesis","previousHash":"","nonce":216,"difficulty":2},{"index":1,"timestamp":"2023-10-27 15:19:12.746","tx":"Mike pays Marty 100
DSCoin","previousHash":"0073cf07aeacb687592854d2b6b6015c15f4dd8c7d64771c52e910413e59e0da","nonce":186,"difficulty":2},{"index":2,"timestamp":"2023-10-27
15:19:31.689","tx":"Marty pays Joe 50
DSCoin","previousHash":"00615d7e516ddb71f730559bd369940cfedfe38450f11279ff09c030ccb277df","nonce":44,"difficulty":2},{"index":3,"timestamp":"2023-10-27 15:19:46.648","tx":"Joe pays Andy 10 DS
Coin","previousHash":"00b69ab8a862c2f5cad161164fb8139dd472cb73ff0f5af0a6f54bf3a4bd4b9d","nonce":296,"difficulty":2}]

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
Mike pays Marty 76 DSCoin
Block 1 now holds Mike pays Marty 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
[{"index":0,"timestamp":"2023-10-27
15:19:00.02","tx":"Genesis","previousHash":"","nonce":216,"difficulty":2},{"index":1,"timestamp":"2023-10-27 15:19:12.746","tx":"Mike pays Marty 76
DSCoin","previousHash":"0073cf07aeacb687592854d2b6b6015c15f4dd8c7d64771c52e910413e59e0da","nonce":186,"difficulty":2},{"index":2,"timestamp":"2023-10-27
15:19:31.689","tx":"Marty pays Joe 50
DSCoin","previousHash":"00615d7e516ddb71f730559bd369940cfedfe38450f11279ff09c030ccb277df","nonce":44,"difficulty":2},{"index":3,"timestamp":"2023-10-27 15:19:46.648","tx":"Joe pays Andy 10 DS
Coin","previousHash":"00b69ab8a862c2f5cad161164fb8139dd472cb73ff0f5af0a6f54bf3a4bd4b9d","nonce":296,"difficulty":2}]

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 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 5 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:
Andy pays Sean 25 DSCoin
Total execution time to add this block was 67 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: 3205
Expected total hashes required for the whole chain: 66560.0
Nonce for the most recent block: 21176
Chain hash: 000032b2bed924b362e7ebe6f4adc57498789e9bc9627173f8886acad1ed1e99

0. View basic blockchain status.

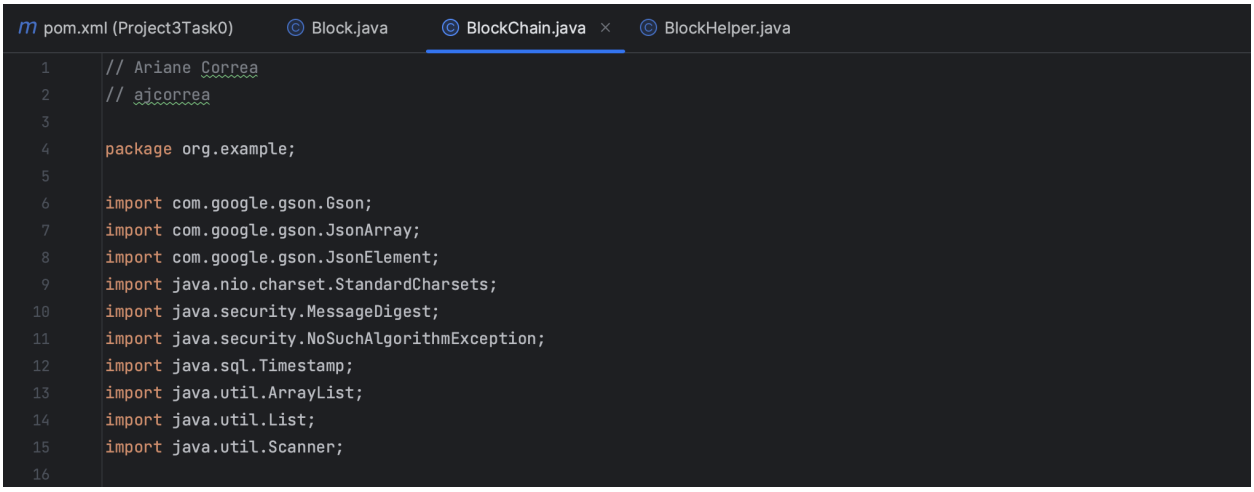
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

1
Enter difficulty > 0
5
Enter transaction:
Ariane pays Marty 100 DSCoin
Total execution time to add this block was 564 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

6
Process finished with exit code 0

Code for BlockChain.java



```java
public class BlockChain {

    // Private instance variables
    28 usages
    private List<Block> blockList;
    6 usages
    private String chainHash;
    3 usages
    private int hashesPerSecond;
    1 usage
    Gson gson = new Gson();

    /**
     * Constructor to initialize a new blockchain.
     */
    1 usage
    public BlockChain() {
        this.blockList = new ArrayList<>();
        this.chainHash = "";
        this.hashesPerSecond = 0;
    }

    /**
     * Get the current blockchain's hash.
     *
     * @return The hash of the entire blockchain.
     */
    3 usages
    public String getChainHash() {
        return chainHash;
    }

    /**
     * Get the current system timestamp.
     *
     * @return A Timestamp object representing the current time.
     */
    10 usages
    public Timestamp getTime() {
        return new Timestamp(System.currentTimeMillis());
    }

    /**
     * Get the most recent block in the blockchain.
     *
     * @return The latest Block object in the chain.
     */
    2 usages
    public Block getLatestBlock() {
        return blockList.get(blockList.size() - 1);
    }

```

```java
    /**
     * Get the size (number of blocks) of the blockchain.
     *
     * @return The number of blocks in the blockchain.
     */
    2 usages
    public int getChainSize() {
        return blockList.size();
    }

    /**
     * Compute the number of hash calculations per second.
     *
     * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
     */
    1 usage
    public void computeHashesPerSecond() throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        int i = 0;
        Timestamp startTime = getTime();
        while (i < 2000000) {
            // Calculate a hash 2,000,000 times with a fixed input
            byte[] encodedHash = md.digest("00000000".getBytes(StandardCharsets.UTF_8));
            i += 1;
        }
        Timestamp endTime = getTime();
        this.hashesPerSecond = 2000000 / (int) (endTime.getTime() - startTime.getTime());
    }

    /**
     * Get the number of hash calculations per second.
     *
     * @return The number of hash calculations per second.
     */
    1 usage
    public int getHashesPerSecond() {
        return this.hashesPerSecond;
    }

    /**
     * Add a new block to the blockchain.
     *
     * @param newBlock The Block object to be added.
     * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
     */
    2 usages
    public void addBlock(Block newBlock) throws NoSuchAlgorithmException {
        if (this.blockList.size() > 0)
            newBlock.setPreviousHash(getChainHash());
        else
            newBlock.setPreviousHash("");
        blockList.add(newBlock);
        this.chainHash = newBlock.proofOfWork();
    }

```

```java
    /**
     * Main method to interact with the blockchain.
     *
     *For difficulty = 2
     * Add block 1: 3 ms
     * Add block 2: 4 ms
     * Add block 3: 3 ms
     * Verify: 2 ms
     * Repair: 17 ms
     *
     * For difficulty = 3
     * Add block 1: 32 ms
     * Add block 2: 33 ms
     * Add block 3: 3 ms
     * Verify: 1 ms
     * Repair: 13 ms
     *
     * For difficulty = 4
     * Add block 1: 156 ms
     * Add block 2: 86 ms
     * Add block 3: 50 ms
     * Verify: 1 ms
     * Repair: 400 ms
     *
     * For difficulty = 5
     * Add block 1: 808 ms
     * Add block 2: 2894 ms
     * Add block 3: 1274 ms
     * Verify: 2 ms
     * Repair: 3507 ms
     *
     * As the difficulty level increases, the time required to add new blocks grows substantially.
     * This is expected because finding a valid nonce becomes more computationally intensive.
     *
     * Verifying the chain remains relatively fast and consistent, regardless of the difficulty level.
     * Verifying the proof of work in existing blocks doesn't become significantly harder as the difficulty increases.
     *
     * Repairing the chain becomes increasingly time-consuming with higher difficulty levels.
     * Recomputing the proof of work for each block, especially for a longer chain, can be a resource-intensive task.
     *
     * The repair operation time is directly affected by the chain length.
     * As the chain grows, the time to repair it also increases.
     *
     * These results illustrate the trade-off between security and performance in a blockchain system.
     * Higher difficulty levels improve security by making it more difficult for malicious actors to alter the chain.
     * However, this increased security comes at the cost of longer block creation and chain repair times.
     *
     * @param args Command-line arguments (not used).
     * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
     */
    public static void main(String args[]) throws NoSuchAlgorithmException {
        // Create a new blockchain instance
        BlockChain chain = new BlockChain();

```

```
166         // Adding genesis block to the chain
167         Block genesis = new Block( index: 0, chain.getTime(),  data: "Genesis",  difficulty: 2);
168         genesis.setPreviousHash("");
169         genesis.proofOfWork();
170         chain.computeHashesPerSecond();
171         chain.addBlock(genesis);
172
173         // Create a scanner for user input
174         Scanner sc = new Scanner(System.in);
175         int option = 0;
176
177         while (true) {
178             System.out.println("0. View basic blockchain status.\n" +
179                     "1. Add a transaction to the blockchain.\n" +
180                     "2. Verify the blockchain.\n" +
181                     "3. View the blockchain.\n" +
182                     "4. Corrupt the chain.\n" +
183                     "5. Hide the corruption by repairing the chain.\n" +
184                     "6. Exit");
185             option = sc.nextInt();
186             switch (option) {
187                 case 0: {
188                     // Status of blockchain
189                     System.out.println("Current size of chain:  " + chain.getChainSize());
190                     System.out.println("Difficulty of most recent block: " + chain.getLatestBlock().getDifficulty());
191                     System.out.println("Total difficulty for all blocks:  " + chain.getTotalDifficulty());
192                     System.out.println("Approximate hashes per second " +
193                             "on this machine: " + chain.getHashesPerSecond());
194                     System.out.println("Expected total hashes required for " +
195                             "the whole chain: " + chain.getTotalExpectedHashes());
196                     System.out.println("Nonce for the most recent block: " + chain.getLatestBlock().getNonce());
197                     System.out.println("Chain hash:  " + chain.getChainHash() + "\n");
198                     break;
199                 }
201                 // Adding block to blockchain
202                 case 1: {
203                     System.out.println("Enter difficulty > 0 \n");
204                     int difficulty = sc.nextInt();
205                     sc.nextLine();
206                     System.out.println("Enter transaction:  \n");
207                     String transaction = sc.nextLine();
208                     Timestamp start = chain.getTime();
209                     // Add new block to the chain
210                     Block newBlock = new Block(chain.getChainSize(), chain.getTime(), transaction, difficulty);
211                     newBlock.setPreviousHash(chain.getChainHash());
212                     newBlock.proofOfWork();
213                     chain.addBlock(newBlock);
214                     Timestamp end = chain.getTime();
215                     System.out.println("Total execution time to add this block was  " +
216                             (end.getTime() - start.getTime()) + " milliseconds");
217                     break;
218                 }
219
```

```
                // Verify blockchain
                case 2: {
                    Timestamp start = chain.getTime();
                    System.out.println("Chain verification: " + chain.isChainValid());
                    Timestamp end = chain.getTime();
                    System.out.println("Total execution time to verify the chain was  " +
                            (end.getTime() - start.getTime()) + " milliseconds");
                    break;
                }

                // View blockchain
                case 3: {
                    System.out.println("View the Blockchain");
                    System.out.println(chain.toString());
                    break;
                }

                // Corrupt blockchain
                case 4: {
                    System.out.println("corrupt the Blockchain");
                    System.out.println("Enter block ID of block to corrupt");
                    int index = sc.nextInt();
                    sc.nextLine();
                    System.out.println("Enter new data for block " + index);
                    String transaction = sc.nextLine();
                    // Setting data to the selected block in the chain
                    chain.getBlock(index).setData(transaction);
                    System.out.println("Block " + index + " now holds " + transaction);
                    break;
                }

                //  Repair blockchain
                case 5: {
                    Timestamp start = chain.getTime();
                    chain.repairChain();
                    Timestamp end = chain.getTime();
                    System.out.println("Total execution time required to repair the chain was " +
                            (end.getTime() - start.getTime()) + " milliseconds");
                    break;
                }

                // Exit the program
                case 6:
                    System.exit( status: 0);
            }
        }
    }

```

```java
    /**
     * Converts the blockchain to a JSON representation.
     *
     * @return JSON string representing the blockchain.
     */
    @Override
    public String toString() {
        JsonArray jsonArray = new JsonArray();
        for (Block b : blockList) {
            jsonArray.add(gson.fromJson(b.toString(), JsonElement.class));
        }
        return jsonArray.toString();
    }

    /**
     * Get a specific block from the blockchain based on its index.
     *
     * @param i The index of the block to retrieve.
     * @return The block at the specified index.
     */
    1 usage
    public Block getBlock(int i) { return blockList.get(i); }

    /**
     * Calculate the total difficulty of all blocks in the blockchain.
     *
     * @return The total difficulty of the blockchain.
     */
    1 usage
    public int getTotalDifficulty() {
        int totalDifficulty = 0;
        for (Block block : blockList) {
            totalDifficulty += block.getDifficulty();
        }
        return totalDifficulty;
    }

    /**
     * Calculate the total expected hash calculations required for the entire blockchain.
     *
     * @return The total expected hash calculations needed for the blockchain.
     */
    1 usage
    public double getTotalExpectedHashes() {
        double totalExpectedHashes = 0;
        for (Block block : blockList)
            totalExpectedHashes += Math.pow(16, block.getDifficulty());  // 16 (16 hex characters) ^ difficulty of block
        return totalExpectedHashes;
    }
```

```java

    /**
     * Check the validity of the blockchain.
     *
     * @return "TRUE" if the blockchain is valid; "FALSE" with an explanation otherwise.
     */
    1 usage
    public String isChainValid() throws NoSuchAlgorithmException {
        // Chain contains only 1 block , i.e. genesis
        if (blockList.size() == 1) {
            Block genesisBlock = this.blockList.get(0);
            String hash = genesisBlock.calculateHash();
            // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
            String prefix = new String(new char[genesisBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
            if (!hash.substring(0, genesisBlock.getDifficulty()).equals(prefix)) {
                return "FALSE \n Improper hash on genesis node";
            } else if (!chainHash.equals(hash)) {
                return "FALSE \n Chain hash and computed hash do not match";
            } else {
                return "TRUE";
            }
        }

        // Chain contains more than 1 block
        if (blockList.size() > 1) {
            for (int i = 1; i < blockList.size(); i++) {
                Block currentBlock = this.blockList.get(i);
                Block previousBlock = this.blockList.get(i - 1);
                String hash = currentBlock.calculateHash();
                String hashPointer = currentBlock.getPreviousHash();
                // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
                String prefix = new String(new char[currentBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
                if (!hash.substring(0, currentBlock.getDifficulty()).equals(prefix))
                    return "FALSE \n Improper hash on node " + i + " Does not begin with " + prefix;
                    // Check proof of work / leading zeros
                else if (!hashPointer.equals(previousBlock.calculateHash()))
                    return "FALSE \n Improper previous hash";
            }
        }

        // A chain hash, check the last element added to to the blocklist
        if (!chainHash.equals(blockList.get(blockList.size() - 1).calculateHash())) {
            return "Chain hash error";
        }

        return "TRUE";
    }

```

```
    1 usage
    public void repairChain() throws NoSuchAlgorithmException {
        // Genesis block
        if (blockList.size() == 1) {
            // Reset previous hash and recompute proof of work
            blockList.get(0).setPreviousHash("");
            blockList.get(0).proofOfWork();
        }

        if (blockList.size() > 1) {
            for (int i = 1; i < blockList.size(); i++) {
                // Reset previous hash and recompute proof of work
                blockList.get(i).setPreviousHash(blockList.get(i - 1).calculateHash());
                blockList.get(i).proofOfWork();
            }

            // Reset chain hash
            this.chainHash = blockList.get(blockList.size() - 1).calculateHash();
        }
    }
}
```

Code for Block.java

pom.xml (Project3Task0) | Block.java | BlockChain.java | BlockHelper.java

```
// Ariane Correa
// ajcorrea

package org.example;

import com.google.gson.JsonObject;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
14 usages
public class Block {

    // Stores the position of the block on the chain
    5 usages
    private int index;
    // Stores the timestamp of the instant when the block was created
    5 usages
    private Timestamp timestamp;
    // Stores single transaction details of the block
    5 usages
    private String data;
    // Stores the SHA256 hash of the block's parent
    4 usages
    private String previousHash;
    // Value determined by POW (Proof of Work) routine
    6 usages
    private BigInteger nonce;
    // The minimum number of leftmost hex digits needed by a proper hash
    7 usages
    private int difficulty;
```

```java
    /**
     * Constructor for creating a Block object.
     *
     * @param index      The position of the block on the chain.
     * @param timestamp  The timestamp of when the block was created.
     * @param data       Single transaction details of the block.
     * @param difficulty The difficulty level for proof of work.
     */
    2 usages
    public Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.difficulty = difficulty;
        this.nonce = BigInteger.ZERO;
    }

    /**
     * Calculate and return the SHA-256 hash of the block.
     *
     * @return The calculated SHA-256 hash.
     * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
     */
    8 usages
    public String calculateHash() throws NoSuchAlgorithmException {
        String parentString = String.valueOf(this.index) + this.timestamp + this.data +
                this.previousHash + this.nonce + this.difficulty;
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] encodedHash = md.digest(parentString.getBytes(StandardCharsets.UTF_8));
        return BlockHelper.bytesToHex(encodedHash);
    }

    /**
     * Get the current nonce value.
     *
     * @return The nonce value.
     */
    1 usage
    public BigInteger getNonce() {
        return nonce;
    }

    /**
     * Get the difficulty level for proof of work.
     *
     * @return The difficulty level.
     */
    7 usages
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Set the difficulty level for proof of work.
     *
     * @param difficulty The new difficulty level.
     */
    no usages
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }
```

```java
    /**
     * Get the position of the block on the chain.
     *
     * @return The block's index.
     */
    no usages
    public int getIndex() {
        return index;
    }

    /**
     * Get the timestamp when the block was created.
     *
     * @return The timestamp of block creation.
     */
    no usages
    public Timestamp getTimestamp() {
        return timestamp;
    }

    /**
     * Get the data (transaction details) of the block.
     *
     * @return The block's data.
     */
    no usages
    public String getData() {
        return data;
    }

    /**
     * Get the SHA-256 hash of the block's parent.
     *
     * @return The SHA-256 hash of the parent block.
     */
    1 usage
    public String getPreviousHash() {
        return previousHash;
    }

    /**
     * Set the position of the block on the chain.
     *
     * @param index The new block index.
     */
    no usages
    public void setIndex(int index) {
        this.index = index;
    }

    /**
     * Set the timestamp when the block was created.
     *
     * @param timestamp The new block creation timestamp.
     */
    no usages
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }

```

```java
    /**
     * Set the data (transaction details) of the block.
     *
     * @param data The new block data.
     */
    1 usage
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Set the SHA-256 hash of the block's parent.
     *
     * @param previousHash The new parent block's hash.
     */
    6 usages
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    /**
     * Convert the block's attributes to a JSON representation.
     *
     * @return A JSON string representing the block's attributes.
     */
    @Override
    public String toString() {
        JsonObject jsonObject = new JsonObject();
        jsonObject.addProperty( property: "index", index);
        jsonObject.addProperty( property: "timestamp", String.valueOf(timestamp));
        jsonObject.addProperty( property: "tx", data);
        jsonObject.addProperty( property: "previousHash", previousHash);
        jsonObject.addProperty( property: "nonce", nonce);
        jsonObject.addProperty( property: "difficulty", difficulty);
        return jsonObject.toString();
    }

    /**
     * Perform Proof of Work (POW) to find a hash that meets the required difficulty level.
     *
     * @return The valid SHA-256 hash after POW.
     * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
     */
    5 usages
    public String proofOfWork() throws NoSuchAlgorithmException {
        String hexHash = calculateHash();
        String matchString = "";
        for (int i = 0; i < this.difficulty; i++) {
            matchString = matchString + "0";
        }
        while (!hexHash.substring(0, this.difficulty).equalsIgnoreCase(matchString)) {
            this.nonce = this.nonce.add(BigInteger.ONE);
            hexHash = calculateHash();
        }
        return hexHash;
    }
}
```

Code for BlockHelper.java

pom.xml (Project3Task0) | Block.java | BlockChain.java | BlockHelper.java

```java
// Ariane Correa
// ajcorrea

package org.example;

1 usage
public class BlockHelper {

    /**
     * Converts a byte array into a hexadecimal string.
     *
     * @param hash The byte array to be converted.
     * @return The hexadecimal string representation of the byte array.
     */

    // Code Reference: https://www.baeldung.com/sha-256-hashing-java
    1 usage
    public static String bytesToHex(byte[] hash) {
        StringBuilder hexString = new StringBuilder( capacity: 2 * hash.length);
        for (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString( i: 0xff & hash[i]);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}

```

95702: Distributed Systems

Project 3

Name: Ariane Correa
Andrew ID: ajcorrea

Task 1

Console Output (Server Side):

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=54059:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project3Task1/target/classes:/Users/arianecorrea/.m2/repository/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.EchoServerTCP
Blockchain server running

We have a visitor
Response :
{"selection":0,"size":1,"chainHash":"0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b06874fa8b","totalHashes":256.0,"totalDiff":2,"recentNonce":387,"diff":2,"hps":2000000}
Adding a block
Setting response to : {"selection":1,"response":"Total execution time to add this block was 11 milliseconds"}
Adding a block
Setting response to : {"selection":1,"response":"Total execution time to add this block was 6 milliseconds"}
Adding a block
Setting response to : {"selection":1,"response":"Total execution time to add this block was 3 milliseconds"}
Verifying entire chain
Chain verification: TRUE
Setting response to: {"selection":2,"response":"Total execution time to verify the chain was 2 milliseconds"}
View the Blockchain
Setting response to: {"selection":3,"response":"[{\"index\":0,\"timestamp\":\"2023-10-27 19:08:10.476\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":387,\"difficulty\":2},{\"index\":1,\"timestamp\":\"2023-10-27 19:08:54.805\",\"tx\":\"Mike pays Marty 100 DSCoin\",\"previousHash\":\"0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b06874fa8b\",\"nonce\":278,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-10-27 19:09:14.847\",\"tx\":\"Marty pays Joe 50

DSCoin\",\"previousHash\":\"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c3fd6545d1\",\"nonce\":355,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-10-27 19:09:29.694\",\"tx\":\"Joe pays Andy 10 DSCoin\",\"previousHash\":\"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398e351a2cc1b\",\"nonce\":236,\"difficulty\":2}]"}
Corrupt the Blockchain
Setting response to: {"selection":4,"response":"Block 1 now holds Mike pays Marty 76 DSCoin"}
View the Blockchain
Setting response to: {"selection":3,"response":"[{\"index\":0,\"timestamp\":\"2023-10-27 19:08:10.476\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":387,\"difficulty\":2},{\"index\":1,\"timestamp\":\"2023-10-27 19:08:54.805\",\"tx\":\"Mike pays Marty 76 DSCoin\",\"previousHash\":\"0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b06874fa8b\",\"nonce\":278,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-10-27 19:09:14.847\",\"tx\":\"Marty pays Joe 50 DSCoin\",\"previousHash\":\"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c3fd6545d1\",\"nonce\":355,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-10-27 19:09:29.694\",\"tx\":\"Joe pays Andy 10 DSCoin\",\"previousHash\":\"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398e351a2cc1b\",\"nonce\":236,\"difficulty\":2}]"}
Verifying entire chain
Chain verification: FALSE
 Improper hash on node 1 Does not begin with 00
Setting response to: {"selection":2,"response":"Total execution time to verify the chain was 2 milliseconds"}
Repairing the entire chain
Setting response to: {"selection":5,"response":"Total execution time required to repair the chain was 11 milliseconds"}
Verifying entire chain
Chain verification: TRUE
Setting response to: {"selection":2,"response":"Total execution time to verify the chain was 2 milliseconds"}
Adding a block
Setting response to : {"selection":1,"response":"Total execution time to add this block was 166 milliseconds"}
Response :
{"selection":0,"size":5,"chainHash":"00006717d60d5f7601c13ca24c43cfef0078dc2ce9cc978ba444b024f9f81a7b","totalHashes":66560.0,"totalDiff":12,"recentNonce":88571,"diff":4,"hps":2000000}
Adding a block
Setting response to : {"selection":1,"response":"Total execution time to add this block was 1674 milliseconds"}

Process finished with exit code 0

Console Output (Client Side):

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=54063:/Applications/IntelliJ
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath
/Users/arianecorrea/IdeaProjects/Project3Task1/target/classes:/Users/arianecorrea/.m2/repos
itory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.EchoClientTCP
The TCP client is running.
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

0
Current size of chain:  1
Difficulty of most recent block: 2
Total difficulty for all blocks:  2
Approximate hashes per second on this machine : 2000000
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block:  387
Chain hash:  "0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b06874fa8b"

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:
Mike pays Marty 100 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

1
Enter difficulty > 0
2
Enter transaction:
Marty pays Joe 50 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:
Joe pays Andy 10 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

2
"Total execution time to verify 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

3
"[{\"index\":0,\"timestamp\":\"2023-10-27
19:08:10.476\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":387,\"difficulty\":2},{\"inde
x\":1,\"timestamp\":\"2023-10-27 19:08:54.805\",\"tx\":\"Mike pays Marty 100
DSCoin\",\"previousHash\":\"0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b
06874fa8b\",\"nonce\":278,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-10-27
19:09:14.847\",\"tx\":\"Marty pays Joe 50
DSCoin\",\"previousHash\":\"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c
3fd6545d1\",\"nonce\":355,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-10-27
19:09:29.694\",\"tx\":\"Joe pays Andy 10
DSCoin\",\"previousHash\":\"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398
e351a2cc1b\",\"nonce\":236,\"difficulty\":2}]"

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
Mike pays Marty 76 DSCoin
"Block 1 now holds Mike pays Marty 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
"[{\"index\":0,\"timestamp\":\"2023-10-27
19:08:10.476\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":387,\"difficulty\":2},{\"inde
x\":1,\"timestamp\":\"2023-10-27 19:08:54.805\",\"tx\":\"Mike pays Marty 76
DSCoin\",\"previousHash\":\"0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b

06874fa8b\",\"nonce\":278,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-10-27 19:09:14.847\",\"tx\":\"Marty pays Joe 50 DSCoin\",\"previousHash\":\"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c3fd6545d1\",\"nonce\":355,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-10-27 19:09:29.694\",\"tx\":\"Joe pays Andy 10 DSCoin\",\"previousHash\":\"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398e351a2cc1b\",\"nonce\":236,\"difficulty\":2}]"

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
"Total execution time to verify 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

5
"Total execution time required to repair 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

2
"Total execution time to verify 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

1
Enter difficulty > 0
4
Enter transaction:
Andy pays Sean 25 DSCoin
"Total execution time to add this block was 166 milliseconds"

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

0
Current size of chain: 5
Difficulty of most recent block: 4
Total difficulty for all blocks: 12
Approximate hashes per second on this machine : 2000000
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 88571
Chain hash: "00006717d60d5f7601c13ca24c43cfef0078dc2ce9cc978ba444b024f9f81a7b"

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
5
Enter transaction:
Ariane pays Marty 100 DSCoin
"Total execution time to add this block was 1674 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

6

Process finished with exit code 0

Code for Block.java

Block.java × BlockChain.java BlockHelper.java EchoClientTCP.java EchoServerTCP.java RequestMessage.java

```java
// Ariane Correa
// ajcorrea

package org.example;

import com.google.gson.JsonObject;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

14 usages
public class Block {
    // Stores the position of the block on the chain
    5 usages
    private int index;
    // Stores the timestamp of the instant when the block was created
    5 usages
    private Timestamp timestamp;
    // Stores single transaction details of the block
    5 usages
    private String data;
    // Stores the SHA256 hash of the block's parent
    4 usages
    private String previousHash;
    // Value determined by POW (Proof of Work) routine
    6 usages
    private BigInteger nonce;
    // The minimum number of leftmost hex digits needed by a proper hash
    7 usages
    private int difficulty;

```

```java
    /**
     * Constructor for creating a Block object.
     *
     * @param index      The position of the block on the chain.
     * @param timestamp  The timestamp of when the block was created.
     * @param data       Single transaction details of the block.
     * @param difficulty The difficulty level for proof of work.
     */
    2 usages
    public Block(int index, Timestamp timestamp, String data, int difficulty) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.difficulty = difficulty;
        this.nonce = BigInteger.ZERO;
    }

    /**
     * Calculate and return the SHA-256 hash of the block.
     *
     * @return The calculated SHA-256 hash.
     * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
     */
    8 usages
    public String calculateHash() throws NoSuchAlgorithmException {
        String parentString = String.valueOf(this.index) + this.timestamp + this.data +
                this.previousHash + this.nonce + this.difficulty;
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] encodedHash = md.digest(
                parentString.getBytes(StandardCharsets.UTF_8));
        return BlockHelper.bytesToHex(encodedHash);
    }

    /**
     * Get the current nonce value.
     *
     * @return The nonce value.
     */
    1 usage
    public BigInteger getNonce() {
        return nonce;
    }

    /**
     * Get the difficulty level for proof of work.
     *
     * @return The difficulty level.
     */
    7 usages
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Simple setter method
     *
     * @param difficulty - determines how much work is required to produce a proper hash
     */
    no usages
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }
```

```java
    /**
     * Get the position of the block on the chain.
     *
     * @return The block's index.
     */
    no usages
    public int getIndex() {
        return index;
    }

    /**
     * Get the timestamp when the block was created.
     *
     * @return The timestamp of block creation.
     */
    no usages
    public Timestamp getTimestamp() {
        return timestamp;
    }

    /**
     * Get the data (transaction details) of the block.
     *
     * @return The block's data.
     */
    no usages
    public String getData() {
        return data;
    }

    /**
     * Get the SHA-256 hash of the block's parent.
     *
     * @return The SHA-256 hash of the parent block.
     */
    1 usage
    public String getPreviousHash() {
        return previousHash;
    }

    /**
     * Set the position of the block on the chain.
     *
     * @param index The new block index.
     */
    no usages
    public void setIndex(int index) {
        this.index = index;
    }

    /**
     * Set the timestamp when the block was created.
     *
     * @param timestamp The new block creation timestamp.
     */
    no usages
    public void setTimestamp(Timestamp timestamp) {
        this.timestamp = timestamp;
    }

```

```java
    /**
     * Set the data (transaction details) of the block.
     *
     * @param data The new block data.
     */
    1 usage
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Set the SHA-256 hash of the block's parent.
     *
     * @param previousHash The new parent block's hash.
     */
    6 usages
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    /**
     * Convert the block's attributes to a JSON representation.
     *
     * @return A JSON string representing the block's attributes.
     */
    @Override
    public String toString() {
        JsonObject jsonObject = new JsonObject();
        jsonObject.addProperty( property: "index", index);
        jsonObject.addProperty( property: "timestamp", String.valueOf(timestamp));
        jsonObject.addProperty( property: "tx", data);
        jsonObject.addProperty( property: "previousHash", previousHash);
        jsonObject.addProperty( property: "nonce", nonce);
        jsonObject.addProperty( property: "difficulty", difficulty);
        return jsonObject.toString();
    }

    /**
     * Perform Proof of Work (POW) to find a hash that meets the required difficulty level.
     *
     * @return The valid SHA-256 hash after POW.
     * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
     */
    5 usages
    public String proofOfWork() throws NoSuchAlgorithmException {

        String hexHash = calculateHash();
        String matchString = "";
        for (int i = 0; i < this.difficulty; i++) {
            matchString = matchString + "0";
        }
        while(!hexHash.substring(0, this.difficulty).equalsIgnoreCase(matchString)){
            this.nonce = this.nonce.add(BigInteger.ONE);
            hexHash = calculateHash();
        }
        return hexHash;
    }
}
```

Code for BlockChain.java

Block.java | BlockChain.java | BlockHelper.java | EchoClientTCP.java | EchoServerTCP.java | RequestMessage.java

```java
// Ariane Correa
// ajcorrea

package org.example;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonElement;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

2 usages
public class BlockChain {

    // Private instance variables
    28 usages
    private List<Block> blockList;
    6 usages
    private String chainHash;
    3 usages
    private int hashesPerSecond;
    1 usage
    Gson gson = new Gson();

    /**
     * Constructor to initialize a new blockchain.
     */
    1 usage
    public BlockChain() {
        this.blockList = new ArrayList<>();
        this.chainHash = "";
        this.hashesPerSecond = 0;
    }

    /**
     * Get the current blockchain's hash.
     *
     * @return The hash of the entire blockchain.
     */
    3 usages
    public String getChainHash() {
        return chainHash;
    }

    /**
     * Get the current system timestamp.
     *
     * @return A Timestamp object representing the current time.
     */
    10 usages
    public Timestamp getTime() { return new Timestamp(System.currentTimeMillis()); }

```

```java
    /**
     * Get the most recent block in the blockchain.
     *
     * @return The latest Block object in the chain.
     */
    2 usages
    public Block getLatestBlock() { return blockList.get(blockList.size() - 1); }

    /**
     * Get the size (number of blocks) of the blockchain.
     *
     * @return The number of blocks in the blockchain.
     */
    2 usages
    public int getChainSize() { return blockList.size(); }

    /**
     * Compute the number of hash calculations per second.
     *
     * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
     */
    1 usage
    public void computeHashesPerSecond() throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        int i = 0;
        Timestamp startTime = getTime();
        while (i < 2000000) {
            // Calculate a hash 2,000,000 times with a fixed input
            byte[] encodedHash = md.digest(
                    "00000000".getBytes(StandardCharsets.UTF_8));
            i += 1;
        }
        Timestamp endTime = getTime();
        this.hashesPerSecond = 2000000 / (endTime.compareTo(startTime));
    }

    /**
     * Get the number of hash calculations per second.
     *
     * @return The number of hash calculations per second.
     */
    1 usage
    public int getHashesPerSecond() { return this.hashesPerSecond; }

    /**
     * Add a new block to the blockchain.
     *
     * @param newBlock The Block object to be added.
     * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
     */
    2 usages
    public void addBlock(Block newBlock) throws NoSuchAlgorithmException {
        if (this.blockList.size() > 0)
            newBlock.setPreviousHash(getChainHash());
        else
            newBlock.setPreviousHash("");
        blockList.add(newBlock);
        this.chainHash = newBlock.proofOfWork();
    }
```

```java
    /**
     * Converts the blockchain to a JSON representation.
     *
     * @return JSON string representing the blockchain.
     */
    @Override
    public String toString() {

        JsonArray jsonArray = new JsonArray();

        for (Block b : blockList) {
            jsonArray.add(gson.fromJson(b.toString(), JsonElement.class));
        }

        return jsonArray.toString();

    }

    /**
     * Get a specific block from the blockchain based on its index.
     *
     * @param i The index of the block to retrieve.
     * @return The block at the specified index.
     */
    1 usage
    public Block getBlock(int i) {
        return blockList.get(i);
    }


    /**
     * Calculate the total difficulty of all blocks in the blockchain.
     *
     * @return The total difficulty of the blockchain.
     */
    1 usage
    public int getTotalDifficulty() {
        int totalDifficulty = 0;
        for (Block block : blockList) {
            totalDifficulty += block.getDifficulty();
        }
        return totalDifficulty;
    }

    /**
     * Calculate the total expected hash calculations required for the entire blockchain.
     *
     * @return The total expected hash calculations needed for the blockchain.
     */
    1 usage
    public double getTotalExpectedHashes() {
        double totalExpectedHashes = 0;
        for (Block block : blockList)
            totalExpectedHashes += Math.pow(16, block.getDifficulty()); // 16 (16 hex characters) ^ difficulty of block
        return totalExpectedHashes;
    }
```

```java
    /**
     * Check the validity of the blockchain.
     *
     * @return "TRUE" if the blockchain is valid; "FALSE" with an explanation otherwise.
     */
    1 usage
    public String isChainValid() throws NoSuchAlgorithmException {
        // Chain contains only 1 block , i.e. genesis
        if (blockList.size() == 1) {
            Block genesisBlock = this.blockList.get(0);
            String hash = genesisBlock.calculateHash();
            // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
            String prefix = new String(new char[genesisBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
            if (!hash.substring(0, genesisBlock.getDifficulty()).equals(prefix)) {
                return "FALSE \n Improper hash on genesis node";
            } else if (!chainHash.equals(hash)) {
                return "FALSE \n Chain hash and computed hash do not match";
            } else {
                return "TRUE";
            }
        }

        // More than 1 block
        if (blockList.size() > 1) {
            for (int i = 1; i < blockList.size(); i++) {
                Block currentBlock = this.blockList.get(i);
                Block previousBlock = this.blockList.get(i - 1);

                String hash = currentBlock.calculateHash();
                String hashPointer = currentBlock.getPreviousHash();
                // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
                String prefix = new String(new char[currentBlock.getDifficulty()]).replace( target: "\0", replacement: "0");

                if (!hash.substring(0, currentBlock.getDifficulty()).equals(prefix))
                    return "FALSE \n Improper hash on node " + i + " Does not begin with " + prefix;
                    // Check proof of work / leading zeros
                else if (!hashPointer.equals(previousBlock.calculateHash()))
                    return "FALSE \n Improper previous hash";
            }
        }

        // Chain hash , check the last element added to to the blocklist
        if (!chainHash.equals(blockList.get(blockList.size() - 1).calculateHash())) {
            return "Chain hash error";
        }

        return "TRUE";
    }
```

```java

    /**
     * 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
     */
    1 usage
    public void repairChain() throws NoSuchAlgorithmException {

        // Genesis block
        if (blockList.size() == 1) {
            //Reset previous hash and recompute proof of work
            blockList.get(0).setPreviousHash("");
            blockList.get(0).proofOfWork();
        }

        if (blockList.size() > 1) {
            for (int i = 1; i < blockList.size(); i++) {
                // Reset previous hash and recompute proof of work
                blockList.get(i).setPreviousHash(blockList.get(i - 1).calculateHash());
                blockList.get(i).proofOfWork();
            }

            // Reset chain hash
            this.chainHash = blockList.get(blockList.size() - 1).calculateHash();
        }
    }
}

```

Code for BlockHelper.java

Block.java | BlockChain.java | BlockHelper.java | EchoClientTCP.java | EchoServerTCP.java | RequestMessage.java

```java
// Ariane Correa
// ajcorrea

package org.example;

1 usage
public class BlockHelper {

    /**
     * Converts a byte array into a hexadecimal string.
     *
     * @param hash The byte array to be converted.
     * @return The hexadecimal string representation of the byte array.
     */

    // Code Reference: https://www.baeldung.com/sha-256-hashing-java
    1 usage
    public static String bytesToHex(byte[] hash) {
        StringBuilder hexString = new StringBuilder( capacity: 2 * hash.length);
        for (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString( i: 0xff & hash[i]);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

Code for EchoClientTCP.java

BlockChain.java | BlockHelper.java | EchoClientTCP.java | EchoServerTCP.java | RequestMessage.java | ResponseMe

```java
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

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

public class EchoClientTCP {
    5 usages
    static Socket clientSocket = null;

    public static void main(String args[]) {
        // Arguments supply hostname
        Scanner sc = new Scanner(System.in);
        try {
            System.out.println("The TCP client is running.");

            clientSocket = new Socket( host: "localhost", port: 7777);
```

```java
        // To read from the server
        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

        // To write to the server
        PrintWriter out = new PrintWriter(new BufferedWriter
                (new OutputStreamWriter(clientSocket.getOutputStream())));

        int option = 0;

        while (true) {

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

            option = sc.nextInt();

            switch (option) {
                // Status of blockchain
                case 0: {
                    RequestMessage requestMessage = new RequestMessage(option);
                    out.println(requestMessage.getRequestJson());
                    out.flush();
                    String reply = in.readLine();
                    // Parsing response object from server as json object
                    JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
                    System.out.println("Current size of chain:  " + jsonReply.get("size"));
                    System.out.println("Difficulty of most recent block: " + jsonReply.get("diff"));
                    System.out.println("Total difficulty for all blocks:  " + jsonReply.get("totalDiff"));
                    System.out.println("Approximate hashes per second on this machine : " + jsonReply.get("hps"));
                    System.out.println("Expected total hashes required for the whole chain: " +
                            jsonReply.get("totalHashes"));
                    System.out.println("Nonce for most recent block:  " + jsonReply.get("recentNonce"));
                    System.out.println("Chain hash:  " + jsonReply.get("chainHash") + "\n");
                    break;
                }

                // Adding block to blockchain
                case 1: {
                    System.out.println("Enter difficulty > 0 \n");
                    int difficulty = sc.nextInt();
                    sc.nextLine();
                    System.out.println("Enter transaction:  \n");
                    String transaction = sc.nextLine();
                    RequestMessage requestMessage = new RequestMessage(option, difficulty, transaction);
                    out.println(requestMessage.getRequestJson());
                    out.flush();
                    String reply = in.readLine();
                    // Parsing response object from server as json object
                    JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
                    System.out.println(jsonReply.get("response"));
                    break;
                }
```

```
83
84
85                      // Verify, View, and Repair blockchain
86                      case 2, 3, 5: {
87
88                          RequestMessage requestMessage = new RequestMessage(option);
89                          out.println(requestMessage.getRequestJson());
90                          out.flush();
91                          String reply = in.readLine();
92                          // Parsing response object from server as json object
93                          JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
94                          System.out.println(jsonReply.get("response"));
95                          break;
96                      }
97
98                      // Corrupt blockchain
99                      case 4: {
100                         System.out.println("corrupt the Blockchain");
101                         System.out.println("Enter block ID of block to corrupt");
102                         int index = sc.nextInt();
103                         sc.nextLine();
104                         System.out.println("Enter new data for block " + index);
105                         String transaction = sc.nextLine();
106                         RequestMessage requestMessage = new RequestMessage(option, index, transaction);
107                         out.println(requestMessage.getRequestJson());
108                         out.flush();
109                         String reply = in.readLine();
110                         //Parsing response object from server as json object
111                         JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
112                         System.out.println(jsonReply.get("response"));
113                         break;
114                     }
115
116                     // Exit the program
117                     case 6:
118                         RequestMessage requestMessage = new RequestMessage(option);
119                         out.println(requestMessage.getRequestJson());
120                         out.flush();
121                         System.exit( status: 0);
122
123
124                 }
125             }
126         } catch (IOException e) {
127             System.out.println("IO Exception:" + e.getMessage());
128         } finally {
129             try {
130                 if (clientSocket != null) {
131                     clientSocket.close();
132                 }
133             } catch (IOException e) {
134                 // Ignore exception on close
135             }
136         }
137     }
138 }
```

Code for EchoServerTCP.java

kChain.java | BlockHelper.java | EchoClientTCP.java | EchoServerTCP.java | RequestMessage.java | ResponseMessage.jav

```java
// Ariane Correa
// ajcorrea

package org.example;

// Code Reference: EchoServerTCP.java from Coulouris text

import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.Scanner;

public class EchoServerTCP {

    public static void main(String args[]) {

        Socket clientSocket = null;

        System.out.println("Blockchain server running");
        try {
            Scanner sc = new Scanner(System.in);
            int serverPort = 7777;  // Read user input
            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.
             */
            clientSocket = listenSocket.accept();
            // If we get here, then we are now connected to a client.

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

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

```

```
            /*
             * Forever,
             *   read a line from the socket
             *   print it to the console
             *   echo it (i.e. write it) back to the client
             */

            BlockChain chain = new BlockChain();
            int option = 0;

            Block genesis = new Block( index: 0, chain.getTime(), data: "Genesis", difficulty: 2);
            genesis.setPreviousHash("");
            genesis.proofOfWork();

            chain.computeHashesPerSecond();
            chain.addBlock(genesis);

            System.out.println("We have a visitor");
            while (true) {
                JsonObject clientReply = new JsonParser().parse(in.nextLine()).getAsJsonObject();
                option = clientReply.get("op").getAsInt();
                switch (option) {
                    // Status of blockchain
                    case 0: {
                        ResponseMessage responseMessage = new ResponseMessage(option,
                                chain.getChainSize(), chain.getLatestBlock().getDifficulty(),
                                chain.getTotalDifficulty(), chain.getHashesPerSecond(), chain.getTotalExpectedHashes(),
                                chain.getLatestBlock().getNonce(), chain.getChainHash());
                        System.out.println("Response : " + responseMessage.getResponseJson());
                        out.println(responseMessage.getResponseJson());
                        out.flush();
                        break;
                    }

                    // Adding block to blockchain
                    case 1: {
                        System.out.println("Adding a block");
                        int difficulty = clientReply.get("difficulty").getAsInt();
                        String transaction = clientReply.get("transaction").getAsString();
                        Timestamp start = chain.getTime();
                        Block newBlock = new Block(chain.getChainSize(), chain.getTime(), transaction, difficulty);
                        newBlock.setPreviousHash(chain.getChainHash());
                        newBlock.proofOfWork();
                        chain.addBlock(newBlock);
                        Timestamp end = chain.getTime();
                        ResponseMessage responseMessage = new ResponseMessage(option, response: "Total execution " +
                                "time to add this block was  " + (end.getTime() - start.getTime()) + " milliseconds");
                        System.out.println("Setting response to : " + responseMessage.getResponseJson());
                        out.println(responseMessage.getResponseJson());
                        out.flush();
                        break;
                    }

```

```
                // Verify blockchain
                case 2: {
                    System.out.println("Verifying entire chain");
                    Timestamp start = chain.getTime();
                    System.out.println("Chain verification: " + chain.isChainValid());
                    Timestamp end = chain.getTime();
                    ResponseMessage responseMessage = new ResponseMessage(option, response: "Total execution " +
                            "time to verify the chain was  " + (end.getTime() - start.getTime()) + " milliseconds");
                    System.out.println("Setting response to: " + responseMessage.getResponseJson());
                    out.println(responseMessage.getResponseJson());
                    out.flush();
                    break;
                }

                // View blockchain
                case 3: {
                    System.out.println("View the Blockchain");
                    ResponseMessage responseMessage = new ResponseMessage(option, chain.toString());
                    System.out.println("Setting response to: " + responseMessage.getResponseJson());
                    out.println(responseMessage.getResponseJson());
                    out.flush();
                    break;

                }

                // Corrupt blockchain
                case 4: {
                    System.out.println("Corrupt the Blockchain");
                    int index = clientReply.get("index").getAsInt();
                    String transaction = clientReply.get("transaction").getAsString();
                    chain.getBlock(index).setData(transaction);
                    ResponseMessage responseMessage = new ResponseMessage(option, response: "Block " + index +
                            " now holds " + transaction);
                    System.out.println("Setting response to: " + responseMessage.getResponseJson());
                    out.println(responseMessage.getResponseJson());
                    out.flush();
                    break;
                }

                //  Repair blockchain
                case 5: {
                    System.out.println("Repairing the entire chain");
                    Timestamp start = chain.getTime();
                    chain.repairChain();
                    Timestamp end = chain.getTime();
                    ResponseMessage responseMessage = new ResponseMessage(option, response: "Total execution " +
                            "time required to repair the chain was " + (end.getTime() - start.getTime()) +
                            " milliseconds");
                    System.out.println("Setting response to: " + responseMessage.getResponseJson());
                    out.println(responseMessage.getResponseJson());
                    out.flush();
                    break;
                }

                // Exit the program
                case 6:
                    System.exit( status: 0);
            }
        }
```

```
            // Handle exceptions
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
            // If quitting (typically by you sending quit signal) clean up sockets
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // Ignore exception on close
            }
        }
    }
}
```

Code for RequestMessage.java

```
kChain.java | BlockHelper.java | EchoClientTCP.java | EchoServerTCP.java | RequestMessage.java | ResponseMessage.ja

// Ariane Correa
// ajcorrea

package org.example;
import com.google.gson.JsonObject;

10 usages
public class RequestMessage {
    6 usages
    private JsonObject requestJson = new JsonObject();

    // options 0, 2, 3, 4, 6
    /**
     * Constructor for single operation
     *
     * @param op
     */
    3 usages
    public RequestMessage(int op) {
        requestJson.addProperty( property: "op", op);
    }

```

```java
    // options 1 , 4
    /**
     * Constructor to send additional property and transaction details
     *
     * @param op
     * @param property
     * @param transaction
     */
    2 usages
    public RequestMessage(int op, int property, String transaction) {
        requestJson.addProperty( property: "op", op);
        if (op == 1)
            requestJson.addProperty( property: "difficulty", property);
        if (op == 4)
            requestJson.addProperty( property: "index", property);
        requestJson.addProperty( property: "transaction", transaction);
    }

    /**
     * Getter method for requestJson
     *
     * @return
     */
    5 usages
    public JsonObject getRequestJson() {
        return requestJson;
    }
}
```

Code for ResponseMessage.java

BlockHelper.java | EchoClientTCP.java | EchoServerTCP.java | RequestMessage.java | ResponseMessage.java

```java
// Ariane Correa
// ajcorrea

package org.example;

import com.google.gson.JsonObject;
import java.math.BigInteger;

12 usages
public class ResponseMessage {
    11 usages
    private JsonObject responseJson = new JsonObject();

```

```java
    // option 0
    /**
     * Constructor for ResponseMessage option 0 with all parameters
     *
     * @param option
     * @param chainSize
     * @param difficulty
     * @param totalDifficulty
     * @param hashesPerSecond
     * @param totalExpectedHashes
     * @param nonce
     * @param chainHash
     */
    1 usage
    public ResponseMessage(int option, int chainSize, int difficulty, int totalDifficulty,
                           int hashesPerSecond, double totalExpectedHashes, BigInteger nonce, String chainHash) {
        responseJson.addProperty( property: "selection", option);
        responseJson.addProperty( property: "size", chainSize);
        responseJson.addProperty( property: "chainHash", chainHash);
        responseJson.addProperty( property: "totalHashes", totalExpectedHashes);
        responseJson.addProperty( property: "totalDiff", totalDifficulty);
        responseJson.addProperty( property: "recentNonce", nonce);
        responseJson.addProperty( property: "diff", difficulty);
        responseJson.addProperty( property: "hps", hashesPerSecond);
    }


    // option 1
    /**
     * Constructor for ResponseMessage with selection message and response
     *
     * @param selection
     * @param response
     */
    5 usages
    public ResponseMessage(int selection , String response) {
        responseJson.addProperty( property: "selection", selection);
        responseJson.addProperty( property: "response", response);
    }


    /**
     * getter method for ResponseJson
     *
     * @return
     */
    12 usages
    public JsonObject getResponseJson() {
        return responseJson;
    }
}
```