Lawrence Hua
Project 3

Task 0 Execution

```
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the curruption by recomputing hashes.
6. Exit.
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Experimented with 2,000,000 hashes.
Approximate hashes per second on this machine: 1266624
Expected total hashes required for the whole chain: 256.000000
Nonce for most recent block: 47
Chain hash: 000d3eab7237a8c5a1f1419915887927256c5917ecdd80afcc22e38c5d6fda31
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 curruption by recomputing hashes.
6. Exit.
1
Enter difficulty > 1
4
Enter transaction
Alice pays Bob 100 DSCoin
Total execution time to add this block was 183 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 curruption by recomputing hashes.
6. Exit.
1
Enter difficulty > 1
4
Enter transaction
```

```
Bob pays Carol 20 DSCoin
Total execution time to add this block was 60 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 curruption by recomputing hashes.
6. Exit.
1
Enter difficulty > 1
4
Enter transaction
Carol pays Donna 10 DSCoin
Total execution time to add this block was 438 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 curruption by recomputing hashes.
6. Exit.
3
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:13:55.701","Tx ":
"Genesis","PrevHash" : "","nonce" : 47,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:14:14.466","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"000d3eab7237a8c5a1f1419915887927256c5917ecdd80afcc22e38c5d6fda31","nonce" :
46581,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:14:22.915","Tx ": "Bob pays Carol 20
DSCoin","PrevHash" :
"000076c61d7d6ca75dc26e19507dd995d3074eb711bd0a7bb6e279056bba00c8","nonce" :
17787,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:14:32.604","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000268da07255f09ab2b1c724d91e24ef54e2247afde9d0391be7aef5a154af","nonce" :
147726,"difficulty": 4}
],"chainHash": "00003cebf88c53691ee75e4d81a794fb59dc9393d147959fe87242fda8405989"}
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 curruption by recomputing hashes.
6. Exit.
2
Verifying entire chain
Chain verification: TRUE
Total execution time required 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 curruption by recomputing hashes.
6. Exit.
4
Enter block ID of block to currupt
2
Enter new data for block 2
Bob pays Tony 30 DSCoin
Block 2 now holds Bob pays Tony 30 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 curruption by recomputing hashes.
6. Exit.
3
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:13:55.701","Tx ":
"Genesis","PrevHash" : "","nonce" : 47,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:14:14.466","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"000d3eab7237a8c5a1f1419915887927256c5917ecdd80afcc22e38c5d6fda31","nonce" :
46581,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:14:22.915","Tx ": "Bob pays Tony 30
DSCoin","PrevHash" :
"000076c61d7d6ca75dc26e19507dd995d3074eb711bd0a7bb6e279056bba00c8","nonce" :
17787,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:14:32.604","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000268da07255f09ab2b1c724d91e24ef54e2247afde9d0391be7aef5a154af","nonce" :
147726,"difficulty": 4}
],"chainHash": "00003cebf88c53691ee75e4d81a794fb59dc9393d147959fe87242fda8405989"}
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 curruption by recomputing hashes.
6. Exit.
2
Verifying entire chain
Chain verification: FALSE
Improper hash on node 2 Does not begin with 0000
Total execution time required 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 curruption by recomputing hashes.
6. Exit.
5
Repairing the entire chain
Total execution time required to repair the chain was 708 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 curruption by recomputing hashes.
6. Exit.
2
Verifying entire chain
Chain verification: TRUE
Total execution time required 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 curruption by recomputing hashes.
6. Exit.
3
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:13:55.701","Tx ":
"Genesis","PrevHash" : "","nonce" : 47,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:14:14.466","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"000d3eab7237a8c5a1f1419915887927256c5917ecdd80afcc22e38c5d6fda31","nonce" :
46581,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:14:22.915","Tx ": "Bob pays Tony 30
DSCoin","PrevHash" :
"000076c61d7d6ca75dc26e19507dd995d3074eb711bd0a7bb6e279056bba00c8","nonce" :
92995,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:14:32.604","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000cc9730b2174da6500c86e294eb47d55e7d329f000b946774ce95a04571df","nonce" :
383740,"difficulty": 4}
],"chainHash": "00008e7c7fd59ac397d867cc7e25b5e9ae45571156651c1010964220e354da9e"}
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 curruption by recomputing hashes.
6. Exit.
6
Exiting...

Process finished with exit code 0
```

Task 0 Block.java

```java
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

public class Block {
   private int index;
   private Timestamp timestamp;
   private String data;
   private String previousHash;
   private BigInteger nonce;
   private int difficulty;
   public int hashes=0;
   public Block(int index, java.sql.Timestamp timestamp, java.lang.String data,
int difficulty) {
       this.index = index;
       this.timestamp = timestamp;
       this.data = data;
       this.previousHash = "";
       this.nonce = BigInteger.ZERO;
       this.difficulty = difficulty;
   }

   //calculte the hash based on input values. return the hash string.
   public java.lang.String calculateHash() {
       String input = index + timestamp.toString() + data + previousHash +
nonce.toString() + difficulty;
       try {
           MessageDigest digest = MessageDigest.getInstance("SHA-256");
           byte[] hash = digest.digest(input.getBytes());
           StringBuilder hexString = new StringBuilder();
           for (byte b : hash) {
               String hex = Integer.toHexString(0xff & b);
               if (hex.length() == 1) hexString.append('0');
               hexString.append(hex);
           }
           return hexString.toString();
       } catch (NoSuchAlgorithmException e) {
           e.printStackTrace();
           return null;
       }
   }

   public BigInteger getNonce() {
       return nonce;
   }

   public String proofOfWork() {
       String hash = calculateHash();
       while (!isValidHash(hash)) {
           hashes++;
           nonce = nonce.add(BigInteger.ONE);
           hash = calculateHash();
       }
       return hash;
   }

   private boolean isValidHash(String hash) {
       for (int i = 0; i < difficulty; i++) {
           if (hash.charAt(i) != '0') {
               return false;
           }
       }
       return true;
   }

   public java.sql.Timestamp getTimestamp() {
       return timestamp;
   }

   public void setTimestamp(java.sql.Timestamp timestamp) {
       this.timestamp = timestamp;
   }

   public java.lang.String getData() {
       return data;
   }

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

   public String getPreviousHash() {
       return previousHash;
   }

   public void setPreviousHash(String previousHash) {
       this.previousHash = previousHash;
   }

   public int getDifficulty() {
       return difficulty;
   }

   public void setDifficulty(int difficulty) {
       this.difficulty = difficulty;
   }

   @Override
   public String toString() {
       return "{" +
               "\"index\": " + index +
               ", \"time stamp \": \"" + timestamp +
               "\", \"Tx \": \"" + data + '\'' +
               "\", \"PrevHash\": \"" + previousHash + '\'' +
               ", \"nonce\": " + nonce +
               ", \"difficulty\": " + difficulty +
               '}';
   }
}
```

Task 0 BlockChain.java

```java
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.*;

/**
* Represents a blockchain consisting of blocks.
*/
public class BlockChain {
   private ArrayList<Block> chain; // List to hold blocks in the blockchain
   private String chainHash; // Hash of the entire blockchain
   private static int hashesPerSecond; // Number of hashes computed per second

   /**
    * Constructs an empty blockchain.
    */
   public BlockChain() {
       this.chain = new ArrayList<>();
       this.chainHash = "";
       this.hashesPerSecond = 0;
   }

   /**
    * Adds a new block to the blockchain.
    * @param newBlock The block to be added.
    */
   public void addBlock(Block newBlock){
       newBlock.setPreviousHash(this.chainHash);
       this.chain.add(newBlock);
       this.chainHash = newBlock.proofOfWork();
   }

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

   /**
    * Gets the number of hashes computed per second.
    * @return The number of hashes computed per second.
```

```java
 */
public int getHashesPerSecond() {
    return hashesPerSecond;
}

/**
 * Retrieves the latest block in the blockchain.
 * @return The latest block.
 */
public Block getLatestBlock() {
    if (!chain.isEmpty()) {
        return chain.get(chain.size() - 1);
    } else {
        return null; // Return null if chain is empty
    }
}

/**
 * Gets the hash of the entire blockchain.
 * @return The hash of the blockchain.
 */
public String getChainHash() {
    if (!chain.isEmpty()) {
        return chainHash;
    } else {
        return ""; // Return empty string if chain is empty
    }
}

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

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

```java
    /**
     * Computes the total expected number of hashes required for the entire blockchain.
     * @return The total expected number of hashes.
     */
    public double getTotalExpectedHashes() {
        long totalExpectedHashes = 0;
        for (Block block : chain) {
            totalExpectedHashes += Math.pow(16, block.getDifficulty());
        }
        return totalExpectedHashes;
    }

    /**
     * Checks if the blockchain is valid.
     * @return "TRUE" if the chain is valid, otherwise an error message.
     */
    public String isChainValid() {
        // Check if the chain is empty
        if (chain.isEmpty()) {
            return "FALSE" + "\n" + "Empty Chain";
        }

        int i = 0;
        while (i < chain.size()) {
            Block block = chain.get(i);
            String computedHash = block.calculateHash();
            String existingHash;
            if (i == chain.size() - 1) {
                existingHash = chainHash; //if theres only 1 block, then assign the chainhash as the existinghash
            } else {
                // Else, use the next block's hash.
                existingHash = chain.get(i + 1).getPreviousHash();
            }

            // Check if chainhash = computedhash or if the difficulty does not match, then false
            int j = 0;
            while (j < block.getDifficulty()) {
                if (computedHash.charAt(j) != '0') {
                    return "FALSE" + "\n" + "Improper hash on node " + i + " Does not begin with " + "0".repeat(block.getDifficulty());
                }
                j++;
            }
            if (!computedHash.equals(existingHash)) {
                return "FALSE" + "\n" + "Improper hash on node " + i + " Does not begin with " + "0".repeat(block.getDifficulty());
            }
            i++;
        }

        // After all checks, return true
        return "TRUE";
    }

    /**
     * Repairs the blockchain by recalculating incorrect hashes.
     */
    public void repairChain() {
        int i = 0;
        while (i < chain.size()) {
            Block block = chain.get(i);
            String computedHash = block.calculateHash();
            String existingHash;

            if (i != chain.size()-1) {
                existingHash = chain.get(i + 1).getPreviousHash();
            } else {
                existingHash = chainHash; //if there's only 1 block, then assign
the chainhash as the existinghash
            }
            while (!computedHash.startsWith("0".repeat(block.getDifficulty())))
{
                block.proofOfWork();
                computedHash = block.calculateHash();
            }
            // If existingHash doesn't equal computed hash
            if (!computedHash.equals(existingHash)) {
                //update the next block's hash if i isnt the same size of the
chain.
                if (i != chain.size()-1){
                    chain.get(i + 1).setPreviousHash(computedHash);
                } else {
                    //assign the chainhash to computed hash if only 1 block
                    chainHash = computedHash;
                }
            }
            i++;
        }
    }


    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        sb.append("\"ds_chain\" : [ ");

        // Iterate through each block in the chain
        for (int i = 0; i < chain.size(); i++) {
            Block block = chain.get(i);
            sb.append("{");
            sb.append("\"index\" : ").append(i).append(",");
            sb.append("\"time stamp\" :
\"").append(block.getTimestamp()).append("\",");
            sb.append("\"Tx \": \"").append(block.getData()).append("\",");
            sb.append("\"PrevHash\" :
\"").append(block.getPreviousHash()).append("\",");
            sb.append("\"nonce\" : ").append(block.getNonce()).append(",");
            sb.append("\"difficulty\":
").append(block.getDifficulty()).append("");
            sb.append("}");
            // Append comma if not last block
            if (i < chain.size() - 1) {
                sb.append(",");
            }
            sb.append("\n");
        }

        sb.append("],");
        sb.append("\"chainHash\": \"").append(chainHash).append("\"");
        sb.append("}");

        return sb.toString();
    }

    /**
     * Computes the number of hashes per second.
     * @throws NoSuchAlgorithmException If the algorithm is not found.
     */
    public static void computeHashesPerSecond() throws NoSuchAlgorithmException
{
        long startTime = System.currentTimeMillis();
        int numberOfHashes = 2000000;
        String str = "00000000";
        for (int i = 0; i < numberOfHashes; i++) {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(str.getBytes());
            md.digest();
        }
        long endTime = System.currentTimeMillis();
        double elapsedTime = ((endTime-startTime)/1000.0);

        hashesPerSecond = (int) (numberOfHashes / elapsedTime);
    }
```

```java
    /**
     * Main method to run the blockchain application.
     * @param args Command-line arguments.
     * @throws NoSuchAlgorithmException If the algorithm is not found.
     */
    public static void main(String[] args) throws NoSuchAlgorithmException {
        BlockChain blockchain = new BlockChain();
        Scanner scanner = new Scanner(System.in);
        Block genesisBlock = new Block(0, blockchain.getTime(), "Genesis", 2);
        blockchain.addBlock(genesisBlock);

        while (true) {
            System.out.println("0. View basic blockchain status.");
            System.out.println("1. Add a transaction to the blockchain.");
            System.out.println("2. Verify the blockchain.");
            System.out.println("3. View the blockchain.");
            System.out.println("4. Corrupt the chain.");
            System.out.println("5. Hide the curruption by recomputing hashes.");
            System.out.println("6. Exit.");
            int choice = scanner.nextInt();
            switch (choice) {
                case 0:
                    System.out.println("Current size of chain: " +
blockchain.getChainSize());
                    System.out.println("Difficulty of most recent block: " +
blockchain.getLatestBlock().getDifficulty());
                    System.out.println("Total difficulty for all blocks: " +
blockchain.getTotalDifficulty());
                    computeHashesPerSecond();
                    System.out.println("Experimented with 2,000,000 hashes.");
                    System.out.println("Approximate hashes per second on this
machine: " + blockchain.getHashesPerSecond());
                    System.out.println("Expected total hashes required for the
whole chain: " + String.format("%.6f", blockchain.getTotalExpectedHashes()));
                    System.out.println("Nonce for most recent block: " +
blockchain.getLatestBlock().getNonce());
                    System.out.println("Chain hash: " +
blockchain.getChainHash());
                    break;
                case 1:
                    System.out.println("Enter difficulty > 1");
                    int difficulty = scanner.nextInt();
                    System.out.println("Enter transaction");
                    scanner.nextLine(); // Consume the newline character
                    String data = scanner.nextLine();
                    Block newBlock = new Block(blockchain.getChainSize(),
blockchain.getTime(), data, difficulty);
                    long startTime = System.currentTimeMillis();
                    blockchain.addBlock(newBlock);
                    long endTime = System.currentTimeMillis();
                    System.out.println("Total execution time to add this block 
was " + (endTime - startTime) + " milliseconds.");
                    break;
                case 2:
                    System.out.println("Verifying entire chain");
                    startTime = System.currentTimeMillis();
                    String validationResult = blockchain.isChainValid();
                    endTime = System.currentTimeMillis();
                    System.out.println("Chain verification: " + 
validationResult);
                    System.out.println("Total execution time required to verify 
the chain was " + (endTime - startTime) + " milliseconds");
                    break;
                case 3:
                    System.out.println("View the Blockchain");
                    System.out.println(blockchain.toString());
                    break;
                case 4:
                    System.out.print("Enter block ID of block to currupt\n");
                    int blockIndex = scanner.nextInt();
                    scanner.nextLine(); // Consume the newline character
                    System.out.println("Enter new data for block "+ blockIndex 
);

                    String newData = scanner.nextLine();
                    blockchain.chain.get(blockIndex).setData(newData);
                    System.out.println("Block " + blockIndex + " now holds " + 
newData);

                    break;
                case 5:
                    System.out.print("Repairing the entire chain\n");

                    startTime = System.currentTimeMillis();
                    blockchain.repairChain();
                    endTime = System.currentTimeMillis();
                    System.out.println("Total execution time required to repair 
the chain was " + (endTime - startTime) + " milliseconds.");
                    break;
                case 6:
                    System.out.println("Exiting...");
                    scanner.close();
                    System.exit(0);
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a number 
between 0 and 6.");
            }
        }
```

```
       /*

       As we increase in difficulty, we see a longer time elapsed in hash
computation due to the
       importance of finding a nonce.

       In my first experiement, I followed the prompt and got the times:

   Test 1
       Approximate hashes per second on this machine: 1529051
       Adding blocks:
       block 1) Total execution time to add this block was 261 milliseconds.
       block 2) Total execution time to add this block was 100 milliseconds.
       block 3) Total execution time to add this block was 166 milliseconds.

       Validating chain:
       before corruption) Total execution time required to verify the chain was
0 milliseconds
       after corruption) Total execution time required to verify the chain was
1 milliseconds

       Repairing:
       Total execution time required to repair the chain was 244 milliseconds.

   Test 2 (increase difficulty to 6)
       Approximate hashes per second on this machine: 1545595
       Adding blocks:
       block 1) Total execution time to add this block was 28272 milliseconds.
       block 2) Total execution time to add this block was 10201 milliseconds.
       block 3) Total execution time to add this block was 26734 milliseconds.

       Validating chain:
       before corruption) Total execution time required to verify the chain was
0 milliseconds
       after corruption) Total execution time required to verify the chain was
1 milliseconds

       Repairing:
       Total execution time required to repair the chain was 105062
milliseconds.

       In conclusion from both tests:
       Adding Blocks: Time taken to add blocks increases with difficulty.
       Validating the Chain: Time taken for validation increases with
difficulty, especially for longer chains with
       more corruption.
       Repairing the Chain: Time taken for repair also increases with
difficulty,
        */
    }
}
```

Task 1 Client Side Execution

```
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the curruption by recomputing hashes.
6. Exit.
Enter your choice:
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: 1539645
Expected total hashes required for the whole chain: 256.000000
Nonce for most recent block: 40
Chain hash: 005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e
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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
1
Enter difficulty > 1:
4
Enter transaction:
Alice pays Bob 100 DSCoin
Total execution time to add this block was 58 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
1
Enter difficulty > 1:
4
Enter transaction:
Bob pays Carol 20 DSCoin
Total execution time to add this block was 113 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
1
Enter difficulty > 1:
4
Enter transaction:
Carol pays Donna 10 DSCoin
Total execution time to add this block was 145 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
3
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:32:39.922","Tx ":
"Genesis","PrevHash" : "","nonce" : 40,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:32:50.72","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e","nonce" :
8989,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:32:59.908","Tx ": "Bob pays Carol 20
DSCoin","PrevHash" :
"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b","nonce" :
40486,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:33:06.848","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000eb9d0a39a854434ec571c9484964edfdc202f6d839b71a0518bc3cb347fe","nonce" :
49983,"difficulty": 4}
],"chainHash":
"0000d8c22d139c77ec90edf351012d4d87a0e1722f2c1fb19a676d4de5129826"}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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
2
Verifying entire chain
Chain verification: TRUE
Total execution time required 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
4
Currupt the Blockchain
Enter block ID of block to corrupt:
2
Enter new data for block 2:
Bob pays Tony 30 DSCoin
Block 2 now holds Bob pays Tony 30 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
3
View the Blockchain
```

```
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:32:39.922","Tx ":
"Genesis","PrevHash" : "","nonce" : 40,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:32:50.72","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e","nonce" :
8989,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:32:59.908","Tx ": "Bob pays Tony 30
DSCoin","PrevHash" :
"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b","nonce" :
40486,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:33:06.848","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000eb9d0a39a854434ec571c9484964edfdc202f6d839b71a0518bc3cb347fe","nonce" :
49983,"difficulty": 4}
],"chainHash":
"0000d8c22d139c77ec90edf351012d4d87a0e1722f2c1fb19a676d4de5129826"}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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
2
Verifying entire chain
Chain verification: FALSE
Improper hash on node 2 Does not begin with 0000
Total execution time required to verify 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
5
Repairing the entire chain
Total execution time required to repair the chain was 284 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
2
Verifying entire chain
Chain verification: TRUE
Total execution time required 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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
3
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:32:39.922","Tx ":
"Genesis","PrevHash" : "","nonce" : 40,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:32:50.72","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e","nonce" :
8989,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:32:59.908","Tx ": "Bob pays Tony 30
DSCoin","PrevHash" :
"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b","nonce" :
178688,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:33:06.848","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000ba452b3b2113c2985e8c64518f6b18859e543198bdb1070755a60f371eaf","nonce" :
73533,"difficulty": 4}
],"chainHash":
"0000f72674c55996dee777f5953f8581c3312ce62521a3b6c834a4b38717fe4d"}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 curruption by recomputing hashes.
6. Exit.
Enter your choice:
6
```

```
Process finished with exit code 0
```

Task 1 Server Side Execution

```
Blockchain server running
We have a visitor
starting to repair
THE JSON REQUEST MESSAGES:
0
1 4 Alice pays Bob 100 DSCoin
1 4 Bob pays Carol 20 DSCoin
1 4 Carol pays Donna 10 DSCoin
3
2
4 2 Bob pays Tony 30 DSCoin
3
2
5
2
3
THE JSON RESPONSE MESSAGES:
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 1539645
Expected total hashes required for the whole chain: 256.000000
Nonce for most recent block: 40
Chain hash: 005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e
{"message":"Current size of chain: 1\nDifficulty of most recent block: 2\nTotal difficulty for all
blocks: 2\nApproximate hashes per second on this machine: 1539645\nExpected total hashes
required for the whole chain: 256.000000\nNonce for most recent block: 40\nChain hash:
005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e\n","numberOfBloc
ks":1}
Total execution time to add this block was 58 milliseconds.
{"message":"Total execution time to add this block was 58 milliseconds.\n","numberOfBlocks":2}
Total execution time to add this block was 113 milliseconds.
{"message":"Total execution time to add this block was 113
milliseconds.\n","numberOfBlocks":3}
Total execution time to add this block was 145 milliseconds.
{"message":"Total execution time to add this block was 145
milliseconds.\n","numberOfBlocks":4}
View the Blockchain
```

```
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:32:39.922","Tx ":
"Genesis","PrevHash" : "","nonce" : 40,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:32:50.72","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e","nonce" :
8989,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:32:59.908","Tx ": "Bob pays Carol 20
DSCoin","PrevHash" :
"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b","nonce" :
40486,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:33:06.848","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000eb9d0a39a854434ec571c9484964edfdc202f6d839b71a0518bc3cb347fe","nonce" :
49983,"difficulty": 4}
],"chainHash":
"0000d8c22d139c77ec90edf351012d4d87a0e1722f2c1fb19a676d4de5129826"}{"message":"Vi
ew the Blockchain\n{\"ds_chain\" : [ {\"index\" : 0,\"time stamp\" : \"2024-03-17
22:32:39.922\",\"Tx \": \"Genesis\",\"PrevHash\" : \"\",\"nonce\" : 40,\"difficulty\": 2},\n{\"index\" :
1,\"time stamp\" : \"2024-03-17 22:32:50.72\",\"Tx \": \"Alice pays Bob 100
DSCoin\",\"PrevHash\" :
\"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e\",\"nonce\" :
8989,\"difficulty\": 4},\n{\"index\" : 2,\"time stamp\" : \"2024-03-17 22:32:59.908\",\"Tx \": \"Bob
pays Carol 20 DSCoin\",\"PrevHash\" :
\"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b\",\"nonce\" :
40486,\"difficulty\": 4},\n{\"index\" : 3,\"time stamp\" : \"2024-03-17 22:33:06.848\",\"Tx \": \"Carol
pays Donna 10 DSCoin\",\"PrevHash\" :
\"0000eb9d0a39a854434ec571c9484964edfdc202f6d839b71a0518bc3cb347fe\",\"nonce\" :
49983,\"difficulty\": 4}\n],\"chainHash\":
\"0000d8c22d139c77ec90edf351012d4d87a0e1722f2c1fb19a676d4de5129826\"}","numberOfBl
ocks":4}
Verifying entire chain
Chain verification: TRUE
Total execution time required to verify the chain was 0 milliseconds
{"message":"Verifying entire chain\nChain verification: TRUE\nTotal execution time required to
verify the chain was 0 milliseconds\n","numberOfBlocks":4}
Block 2 now holds Bob pays Tony 30 DSCoin
{"message":"Block 2 now holds Bob pays Tony 30 DSCoin\n","numberOfBlocks":4}
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:32:39.922","Tx ":
"Genesis","PrevHash" : "","nonce" : 40,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:32:50.72","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e","nonce" :
8989,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:32:59.908","Tx ": "Bob pays Tony 30
DSCoin","PrevHash" :
"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b","nonce" :
40486,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:33:06.848","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000eb9d0a39a854434ec571c9484964edfdc202f6d839b71a0518bc3cb347fe","nonce" :
49983,"difficulty": 4}
],"chainHash":
"0000d8c22d139c77ec90edf351012d4d87a0e1722f2c1fb19a676d4de5129826"}{"message":"Vi
ew the Blockchain\n{\"ds_chain\" : [ {\"index\" : 0,\"time stamp\" : \"2024-03-17
22:32:39.922\",\"Tx \": \"Genesis\",\"PrevHash\" : \"\",\"nonce\" : 40,\"difficulty\": 2},\n{\"index\" :
1,\"time stamp\" : \"2024-03-17 22:32:50.72\",\"Tx \": \"Alice pays Bob 100
DSCoin\",\"PrevHash\" :
\"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e\",\"nonce\" :
8989,\"difficulty\": 4},\n{\"index\" : 2,\"time stamp\" : \"2024-03-17 22:32:59.908\",\"Tx \": \"Bob
pays Tony 30 DSCoin\",\"PrevHash\" :
\"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b\",\"nonce\" :
40486,\"difficulty\": 4},\n{\"index\" : 3,\"time stamp\" : \"2024-03-17 22:33:06.848\",\"Tx \": \"Carol
pays Donna 10 DSCoin\",\"PrevHash\" :
\"0000eb9d0a39a854434ec571c9484964edfdc202f6d839b71a0518bc3cb347fe\",\"nonce\" :
49983,\"difficulty\": 4}\n],\"chainHash\":
\"0000d8c22d139c77ec90edf351012d4d87a0e1722f2c1fb19a676d4de5129826\"}","numberOfBl
ocks":4}
Verifying entire chain
Chain verification: FALSE
Improper hash on node 2 Does not begin with 0000
Total execution time required to verify the chain was 5 milliseconds
{"message":"Verifying entire chain\nChain verification: FALSE\nImproper hash on node 2 Does
not begin with 0000\nTotal execution time required to verify the chain was 5
milliseconds\n","numberOfBlocks":4}
Repairing the entire chain
Total execution time required to repair the chain was 284 milliseconds.
{"message":"Repairing the entire chain\nTotal execution time required to repair the chain was
284 milliseconds.\n","numberOfBlocks":4}
Verifying entire chain
Chain verification: TRUE
Total execution time required to verify the chain was 0 milliseconds
{"message":"Verifying entire chain\nChain verification: TRUE\nTotal execution time required to
verify the chain was 0 milliseconds\n","numberOfBlocks":4}
View the Blockchain
{"ds_chain" : [ {"index" : 0,"time stamp" : "2024-03-17 22:32:39.922","Tx ":
"Genesis","PrevHash" : "","nonce" : 40,"difficulty": 2},
{"index" : 1,"time stamp" : "2024-03-17 22:32:50.72","Tx ": "Alice pays Bob 100
DSCoin","PrevHash" :
"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e","nonce" :
8989,"difficulty": 4},
{"index" : 2,"time stamp" : "2024-03-17 22:32:59.908","Tx ": "Bob pays Tony 30
DSCoin","PrevHash" :
"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b","nonce" :
178688,"difficulty": 4},
{"index" : 3,"time stamp" : "2024-03-17 22:33:06.848","Tx ": "Carol pays Donna 10
DSCoin","PrevHash" :
"0000ba452b3b2113c2985e8c64518f6b18859e543198bdb1070755a60f371eaf","nonce" :
73533,"difficulty": 4}
],"chainHash":
"0000f72674c55996dee777f5953f8581c3312ce62521a3b6c834a4b38717fe4d"}{"message":"Vie
w the Blockchain\n{\"ds_chain\" : [ {\"index\" : 0,\"time stamp\" : \"2024-03-17 22:32:39.922\",\"Tx
\": \"Genesis\",\"PrevHash\" : \"\",\"nonce\" : 40,\"difficulty\": 2},\n{\"index\" : 1,\"time stamp\" :
\"2024-03-17 22:32:50.72\",\"Tx \": \"Alice pays Bob 100 DSCoin\",\"PrevHash\" :
\"005fa289911d85459c9e23e414d7c85f2ff5959921644d0365e18a614742d04e\",\"nonce\" :
8989,\"difficulty\": 4},\n{\"index\" : 2,\"time stamp\" : \"2024-03-17 22:32:59.908\",\"Tx \": \"Bob
pays Tony 30 DSCoin\",\"PrevHash\" :
\"0000116280b54b4978fa0f3b1accac57e2f99859fdcfaf856e3b40f948e21d0b\",\"nonce\" :
178688,\"difficulty\": 4},\n{\"index\" : 3,\"time stamp\" : \"2024-03-17 22:33:06.848\",\"Tx \":
\"Carol pays Donna 10 DSCoin\",\"PrevHash\" :
\"0000ba452b3b2113c2985e8c64518f6b18859e543198bdb1070755a60f371eaf\",\"nonce\" :
73533,\"difficulty\": 4}\n],\"chainHash\":
\"0000f72674c55996dee777f5953f8581c3312ce62521a3b6c834a4b38717fe4d\"}","numberOfBl
ocks":4}
Number of Blocks on Chain == 4
```

Task 1 Client Source Code in the file ClientTCP.java.

Client code

```java
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class ClientTCP {
   public static void main(String args[]) {
       Socket clientSocket = null;
```

```
        Scanner scanner = new Scanner(System.in);

        try {
            int serverPort = 7777;
            clientSocket = new Socket("localhost", serverPort);

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

            String userInput;
            while (true) {
                // Display the menu
                displayMenu();

                // Get user input
                userInput = scanner.nextLine();

                // If user chooses to exit
                if (userInput.equals("6")) {
                    // Send exit signal to server
                    out.println(userInput);
                    out.flush();
                    break; // Exit the loop
                }
                // If user chose option 1, which requires more interaction
                if (userInput.equals("1")) {
                    // Request additional information from the user
                    System.out.println("Enter difficulty > 1: ");
                    String difficultyInput = scanner.nextLine();
                    System.out.println("Enter transaction: ");
                    String transactionInput = scanner.nextLine();
                    RequestMessage requestMessage = new
RequestMessage(userInput, difficultyInput, transactionInput);
                    // Convert the request to JSON
                    String jsonRequest = requestMessage.toJson();

                    // Send the JSON request to the server
                    out.println(jsonRequest);
                    out.flush();
                }
                // If user chose option 4, which requires more interaction
                else if (userInput.equals("4")) {
                    System.out.println("Currupt the Blockchain");
                    System.out.println("Enter block ID of block to corrupt: ");
                    String blockID = scanner.nextLine();
                    System.out.println("Enter new data for block " + blockID +
": ");
```

```
                    String newData = scanner.nextLine();
                    RequestMessage requestMessage = new
RequestMessage(userInput,blockID, newData);

                    // Convert the request to JSON
                    String jsonRequest = requestMessage.toJson();

                    // Send the JSON request to the server
                    out.println(jsonRequest);
                    out.flush();
                }
                // For other cases, send user input directly to server
                else {
                    RequestMessage requestMessage = new
RequestMessage(userInput);
                    // Convert the request to JSON
                    String jsonRequest = requestMessage.toJson();

                    // Send the JSON request to the server
                    out.println(jsonRequest);
                    out.flush();
                }

                // Read response from server
                String jsonResponse = in.readLine();

                // Deserialize the JSON response into a response object
                ResponseMessage responseMessage =
ResponseMessage.fromJSON(jsonResponse);

// Extract the content from the response message
                String content = responseMessage.getMessage();

// Print the content
                System.out.print(content);
            }
        } catch (IOException e) {
            System.out.println("IO Exception: " + e.getMessage());
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }
```

```
   private static void displayMenu() {
       System.out.println("0. View basic blockchain status.");
       System.out.println("1. Add a transaction to the blockchain.");
       System.out.println("2. Verify the blockchain.");
       System.out.println("3. View the blockchain.");
       System.out.println("4. Corrupt the chain.");
       System.out.println("5. Hide the curruption by recomputing hashes.");
       System.out.println("6. Exit.");
       System.out.println("Enter your choice: ");
   }
}
```

RequestMessage Class

```
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import com.google.gson.Gson;

public class RequestMessage {
   private String message;
   private int numberOfBlocks; // Number of blocks in the blockchain

   public RequestMessage(String message) {
       this.message = message;
       this.numberOfBlocks = numberOfBlocks;

   }

   public RequestMessage(String userInput, String difficultyInput, String
transactionInput) {
       this.message = userInput + " " + difficultyInput + " " +
transactionInput;
   }

   public String getMessage() {
       return message;
   }

   public String toJson() {
       Gson gson = new Gson();
       return gson.toJson(this);
   }

   public static RequestMessage fromJson(String json) {
       Gson gson = new Gson();
       return gson.fromJson(json, RequestMessage.class);
   }
}
```

ResponseMessage class

```java
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import com.google.gson.Gson;

public class ResponseMessage {
   private String message; // Message to be sent to the client
   private int numberOfBlocks; // Number of blocks in the blockchain

   // Constructor
   public ResponseMessage(String message, int numberOfBlocks) {
       this.message = message;
       this.numberOfBlocks = numberOfBlocks;
   }

   public ResponseMessage() {

   }

   // Getters and Setters
   public String getMessage() {
       return message;
   }

   public void setMessage(String message) {
       this.message = message;
   }

   public int getNumberOfBlocks() {
       return numberOfBlocks;
   }

   public void setNumberOfBlocks(int numberOfBlocks) {
       this.numberOfBlocks = numberOfBlocks;
   }

   // Method to serialize the object to a JSON string
   public String toJSON() {
       Gson gson = new Gson();
       return gson.toJson(this);
   }

   // Method to deserialize a JSON string into an object
```

```
    public static ResponseMessage fromJSON(String json) {
        Gson gson = new Gson();
        return gson.fromJson(json, ResponseMessage.class);
    }

}
```

Task 1 Server Source Code in the file ServerTCP.java.

Server code:

```
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import com.google.gson.JsonSyntaxException;

import java.io.*;
import java.net.*;
import java.security.NoSuchAlgorithmException;

public class ServerTCP {
    // Initialize a blockchain instance
    public static BlockChain blockchain = new BlockChain();

    // StringBuilder to accumulate JSON request messages
    private static StringBuilder requestMessages = new StringBuilder();

    // StringBuilder to accumulate JSON response messages
    private static StringBuilder responseMessages = new StringBuilder();

    public static void main(String args[]) {
        ServerSocket serverSocket = null;
        Socket clientSocket = null;
        try {
            int serverPort = 7777;
            serverSocket = new ServerSocket(serverPort);

            // Server started message
            System.out.println("Blockchain server running");

            while (true) {
                // Accept incoming connections
                clientSocket = serverSocket.accept();

                // Initialize input and output streams
                BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

```
               PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));

               // Notify that a visitor has connected
               System.out.println("We have a visitor");

               // Create the genesis block for the blockchain
               Block genesisBlock = new Block(0, blockchain.getTime(),
"Genesis", 2);
               blockchain.addBlock(genesisBlock);

               String request;
               while ((request = in.readLine()) != null) {
                   try {
                       if (request.equals("6")) {
                           System.out.println("THE JSON REQUEST MESSAGES:");
                           System.out.print(requestMessages);
                           System.out.println("THE JSON RESPONSE MESSAGES:");
                           System.out.print(responseMessages);

                           // Output total number of blocks on the chain
                           System.out.println("Number of Blocks on Chain == " +
blockchain.getChainSize());
                           System.out.println();
                           break;
                       }
                       // Deserialize the JSON request into a RequestMessage
object
                       RequestMessage requestMessage =
RequestMessage.fromJson(request);

                       // Extract the message from the request
                       String userInput = requestMessage.getMessage();

                       // Process the request and generate a response
                       String response = processRequest(userInput);

                       // Accumulate the JSON request message
                       requestMessages.append(userInput).append("\n");

                       // Serialize the response object to JSON
                       ResponseMessage responseMessage = new
ResponseMessage(response, blockchain.getChainSize());
                       String jsonResponse = responseMessage.toJSON();

                       // Accumulate the JSON response message
                       responseMessages.append(jsonResponse).append("\n");

                       // Send the JSON response to the client
```

```java
                        out.println(jsonResponse);
                        out.flush();

                        // Break the loop if the user chose to exit

                    } catch (JsonSyntaxException e) {
                        // Handle JSON parsing error
                        System.out.println("Invalid JSON request: " +
e.getMessage());
                        // Optionally, you can send an error response to the
client
                        out.println("Invalid JSON request: " + e.getMessage());
                        out.flush();
                    }
                }
            }
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            try {
                // Close server socket
                if (serverSocket != null) {
                    serverSocket.close();
                }
                // Close client socket
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    // Method to process the request and generate a response
    private static String processRequest(String request) {
        String[] parts = request.split(" ", 3);
        int choice = Integer.parseInt(parts[0]);
        StringBuilder response = new StringBuilder();
        switch (choice) {
            case 0:
                response.append(getBasicBlockchainStatus());
                break;
            case 1:
                response.append(addTransaction(parts[2],
Integer.parseInt(parts[1])));
                break;
            case 2:
                response.append(verifyBlockchain());
                break;
            case 3:
                response.append(viewBlockchain());
                break;
            case 4:
                response.append(corruptChain(Integer.parseInt(parts[1]),
parts[2]));
                break;
            case 5:
                response.append(repairChain());
                break;
            case 6:
                response.append("Exiting...").append("\n");
                break;
            default:
                response.append("Invalid choice").append("\n");
                break;
        }
        responseMessages.append(response);
        return response.toString();
    }



    // Method to get basic blockchain status
    private static String getBasicBlockchainStatus() {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        response.append("Current size of chain:
").append(blockchain.getChainSize()).append("\n");
        response.append("Difficulty of most recent block:
").append(blockchain.getLatestBlock().getDifficulty()).append("\n");
        response.append("Total difficulty for all blocks:
").append(blockchain.getTotalDifficulty()).append("\n");
        try {
            // Attempt to compute hashes per second
            BlockChain.computeHashesPerSecond();
            response.append("Approximate hashes per second on this machine:
").append(blockchain.getHashesPerSecond()).append("\n");
        } catch (NoSuchAlgorithmException e) {
            // If an exception occurs, include error message in the response
            response.append("Failed to compute hashes per second:
").append(e.getMessage()).append("\n");
        }
        response.append("Expected total hashes required for the whole chain:
").append(String.format("%.6f",
blockchain.getTotalExpectedHashes())).append("\n");
        response.append("Nonce for most recent block:
").append(blockchain.getLatestBlock().getNonce()).append("\n");
```

```
        response.append("Chain hash:
").append(blockchain.getChainHash()).append("\n");
        return response.toString();
    }

    // Method to add a transaction to the blockchain
    private static String addTransaction(String data, int difficulty) {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        String newData = data;
        Block newBlock = new Block(blockchain.getChainSize(),
blockchain.getTime(), newData, difficulty);
        long startTime = System.currentTimeMillis();
        blockchain.addBlock(newBlock);
        long endTime = System.currentTimeMillis();
        response.append("Total execution time to add this block was
").append(endTime - startTime).append(" milliseconds.").append("\n");
        return response.toString();
    }

    // Method to verify the blockchain
    private static String verifyBlockchain() {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        response.append("Verifying entire chain\n");
        long startTime = System.currentTimeMillis();
        String validationResult = blockchain.isChainValid();
        long endTime = System.currentTimeMillis();
        response.append("Chain verification:
").append(validationResult).append("\n");
        response.append("Total execution time required to verify the chain was
").append(endTime - startTime).append(" milliseconds").append("\n");
        return response.toString();
    }

    // Method to view the blockchain
    private static String viewBlockchain() {
        return "View the Blockchain\n" + blockchain.toString();
    }

    // Method to corrupt the blockchain
    private static String corruptChain(int blockIndex, String newData) {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        blockchain.chain.get(blockIndex).setData(newData);
        response.append("Block ").append(blockIndex).append(" now holds
").append(newData).append("\n");;
        return response.toString();
    }
```

```
   // Method to repair the blockchain
   private static String repairChain() {
       System.out.println("starting to repair");
       // StringBuilder to accumulate response
       StringBuilder response = new StringBuilder();
       response.append("Repairing the entire chain\n");
       long startTime = System.currentTimeMillis();
       blockchain.repairChain();
       long endTime = System.currentTimeMillis();
       response.append("Total execution time required to repair the chain was
").append(endTime - startTime).append(" milliseconds.").append("\n");
       return response.toString();
   }

}
```

ResponseMessage class:

```
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import com.google.gson.Gson;

public class ResponseMessage {
   private String message; // Message to be sent to the client
   private int numberOfBlocks; // Number of blocks in the blockchain

   // Constructor
   public ResponseMessage(String message, int numberOfBlocks) {
       this.message = message;
       this.numberOfBlocks = numberOfBlocks;
   }

   public ResponseMessage() {

   }

   // Getters and Setters
   public String getMessage() {
       return message;
   }

   public void setMessage(String message) {
       this.message = message;
   }

   public int getNumberOfBlocks() {
       return numberOfBlocks;
   }

   public void setNumberOfBlocks(int numberOfBlocks) {
       this.numberOfBlocks = numberOfBlocks;
   }

   // Method to serialize the object to a JSON string
   public String toJSON() {
       Gson gson = new Gson();
       return gson.toJson(this);
   }

   // Method to deserialize a JSON string into an object
   public static ResponseMessage fromJSON(String json) {
       Gson gson = new Gson();
       return gson.fromJson(json, ResponseMessage.class);
   }

}
```

RequestMessage Class:

```
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import com.google.gson.Gson;

public class RequestMessage {
    private String message;
    private int numberOfBlocks; // Number of blocks in the blockchain

    public RequestMessage(String message) {
        this.message = message;
        this.numberOfBlocks = numberOfBlocks;

    }

    public RequestMessage(String userInput, String difficultyInput, String
transactionInput) {
        this.message = userInput + " " + difficultyInput + " " +
transactionInput;
    }

    public String getMessage() {
        return message;
    }

    public String toJson() {
        Gson gson = new Gson();
        return gson.toJson(this);
    }

    public static RequestMessage fromJson(String json) {
        Gson gson = new Gson();
        return gson.fromJson(json, RequestMessage.class);
    }
}
```

Project3Task2SigningClient

```
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.Scanner;

public class SigningClientTCP {

    private static BigInteger RSAPublicKey;
    private static BigInteger RSAPrivateKey;
    private static BigInteger RSAModulus;
    private static BigInteger clientID;

    public static void main(String args[]) {
        Socket clientSocket = null;
        Scanner scanner = new Scanner(System.in);

        try {
            int serverPort = 7777;
            clientSocket = new Socket("localhost", serverPort);

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
```

```
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));


            Map<String, BigInteger> rsaKeys = createRSAPubAndPrivKeys(); //
Generate RSA keys
            RSAPublicKey = rsaKeys.get("publicExponent");
            RSAPrivateKey = rsaKeys.get("privateKey");
            RSAModulus = rsaKeys.get("modulus");
            clientID = computeClientID(RSAPublicKey,RSAModulus); // Compute
client ID

            System.out.println(clientID);
            System.out.println("RSA Public Key (e,n): " + RSAPublicKey + "," +
RSAModulus);
            System.out.println("RSA Private Key (d,n): " + RSAPrivateKey + "," +
RSAModulus);
            System.out.println("Client ID: " + clientID);
            while (true) {
                displayMenu();

                String userInput = scanner.nextLine();

                if (userInput.equals("6")) {
                    out.println(userInput);
                    out.flush();
                    break;
                }

                if (userInput.equals("1")) {
                    System.out.println("Enter difficulty > 1: ");
                    String difficultyInput = scanner.nextLine();
                    System.out.println("Enter transaction: ");
                    String transactionInput = scanner.nextLine();

                    String messageToSign = userInput + clientID;
                    String signature = sign(messageToSign);

                    RequestMessage requestMessage = new
RequestMessage(RSAModulus, clientID.toString(), userInput, difficultyInput,
transactionInput, RSAPublicKey, signature);

                    String jsonRequest = requestMessage.toJson();

                    out.println(jsonRequest);
                    out.flush();
                } else if (userInput.equals("4")) {
                    System.out.println("Corrupt the Blockchain");
                    System.out.println("Enter block ID of block to corrupt: ");
                    String blockID = scanner.nextLine();
                    System.out.println("Enter new data for block " + blockID +
": ");
                    String newData = scanner.nextLine();

                    String messageToSign = userInput + clientID;
                    String signature = sign(messageToSign);

                    RequestMessage requestMessage = new
RequestMessage(RSAModulus, clientID.toString(), userInput, blockID, newData,
RSAPublicKey, signature);

                    String jsonRequest = requestMessage.toJson();

                    out.println(jsonRequest);
                    out.flush();
                } else {
                    String messageToSign = userInput + clientID;
                    String signature = sign(messageToSign);
                    System.out.println("this is sig@@" + signature);
                    RequestMessage requestMessage = new
RequestMessage(RSAModulus,userInput, clientID.toString(), RSAPublicKey,
signature);

                    String jsonRequest = requestMessage.toJson();

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

                String jsonResponse = in.readLine();
                System.out.println("This is the response" + jsonResponse);
                ResponseMessage responseMessage =
ResponseMessage.fromJSON(jsonResponse);

                String content = responseMessage.getMessage();

                System.out.print(content);
            }
        } catch (IOException e) {
            System.out.println("IO Exception: " + e.getMessage());
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    private static void displayMenu() {
        System.out.println("0. View basic blockchain status.");
        System.out.println("1. Add a transaction to the blockchain.");
        System.out.println("2. Verify the blockchain.");
        System.out.println("3. View the blockchain.");
        System.out.println("4. Corrupt the chain.");
        System.out.println("5. Hide the corruption by recomputing hashes.");
        System.out.println("6. Exit.");
        System.out.println("Enter your choice: ");
    }

    private static Map<String, BigInteger> createRSAPubAndPrivKeys() {
        Random rnd = new Random();
        BigInteger p = new BigInteger(400, 100, rnd);
        BigInteger q = new BigInteger(400, 100, rnd);
        BigInteger n = p.multiply(q);
        BigInteger e = new BigInteger("65537");
        BigInteger phi =
(p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
        BigInteger d = e.modInverse(phi);

        Map<String, BigInteger> keys = new HashMap<>();
        keys.put("publicExponent", e);
        keys.put("modulus", n);
        keys.put("privateKey", d);
        return keys;
    }

    private static BigInteger computeClientID(BigInteger RSAPublicKey,
BigInteger RSAModulus) {
        String idString = RSAPublicKey.toString() + RSAModulus.toString();
        System.out.println("id stromg" + idString);
        return computeHash(idString);
    }

    private static BigInteger computeHash(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(input.getBytes());
            return new BigInteger(1, hash);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return BigInteger.ZERO;
        }
    }

    public static String sign(String message) throws Exception {

        // compute the digest with SHA-256
        byte[] bytesOfMessage = message.getBytes("UTF-8");
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] bigDigest = md.digest(bytesOfMessage);

        // we only want two bytes of the hash for ShortMessageSign
        // we add a 0 byte as the most significant byte to keep
        // the value to be signed non-negative.
        byte[] messageDigest = new byte[3];
        messageDigest[0] = 0;   // most significant set to 0
        messageDigest[1] = bigDigest[0]; // take a byte from SHA-256
        messageDigest[2] = bigDigest[1]; // take a byte from SHA-256

        // The message digest now has three bytes. Two from SHA-256
        // and one is 0.

        // From the digest, create a BigInteger
        BigInteger m = new BigInteger(messageDigest);

        // encrypt the digest with the private key
        BigInteger c = m.modPow(RSAPrivateKey,RSAModulus);

        // return this as a big integer string
        return c.toString();
    }
    private static byte[] computeSHA256Hash(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            return digest.digest(data.getBytes());
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }



}
```

Project3Task2VerifyingServer

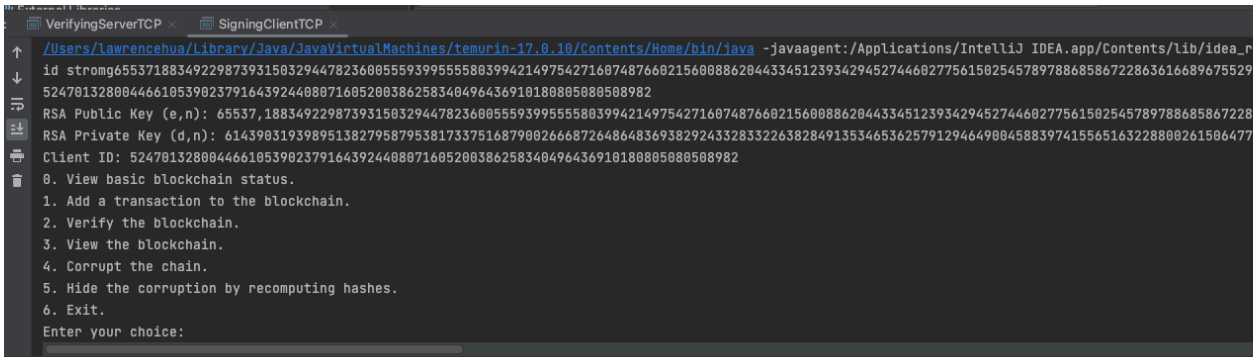
```java
package cmu.edu.ds;
//lawrence hua
//lhua
//lhua@andrew.cmu.edu
import com.google.gson.JsonSyntaxException;

import java.io.*;
import java.math.BigInteger;
import java.net.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class VerifyingServerTCP {
   // Initialize a blockchain instance
   public static BlockChain blockchain = new BlockChain();

   // StringBuilder to accumulate JSON request messages
   private static StringBuilder requestMessages = new StringBuilder();

   // StringBuilder to accumulate JSON response messages
   private static StringBuilder responseMessages = new StringBuilder();

   public static void main(String args[]) {
       ServerSocket serverSocket = null;
       Socket clientSocket = null;
       try {
           int serverPort = 7777;
           serverSocket = new ServerSocket(serverPort);

           // Server started message
           System.out.println("Blockchain server running");

           while (true) {
               // Accept incoming connections
               clientSocket = serverSocket.accept();

               // Initialize input and output streams
               BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
               PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));

               // Notify that a visitor has connected
               System.out.println("We have a visitor");

               // Create the genesis block for the blockchain
               Block genesisBlock = new Block(0, blockchain.getTime(),
"Genesis", 2);
               blockchain.addBlock(genesisBlock);

               String request;
               while ((request = in.readLine()) != null) {
                   try {
                       if (request.equals("6")) {
                           System.out.println("THE JSON REQUEST MESSAGES:");
                           System.out.print(requestMessages);
                           System.out.println("THE JSON RESPONSE MESSAGES:");
                           System.out.print(responseMessages);

                           // Output total number of blocks on the chain
                           System.out.println("Number of Blocks on Chain == " +
blockchain.getChainSize());
                           System.out.println();
                           break;
                       }

                       // Deserialize the JSON request into a RequestMessage
object
                       RequestMessage requestMessage =
RequestMessage.fromJson(request);

                       // Extract client ID, public key, request, and signature
from the request
                       String clientID = requestMessage.getClientID();
                       String publicKey = requestMessage.getRSAPublicKey();
                       String message = requestMessage.getMessage();
                       String signature =
String.valueOf(requestMessage.getSignature());
                       BigInteger modulus = requestMessage.getModulus();
                       System.out.println(modulus);

                       // Verify if public key hashes to the provided client ID
                       if (!verifyClientID(publicKey,modulus, clientID)) {
                           out.println("Error in request: Public key does not
hash to the provided ID.");
                           out.flush();
                           continue;
                       }

                       // Verify the signature of the request
                       if (!verifySignature( message,
signature,publicKey,modulus)) {
                           out.println("Error in request: Signature
verification failed.");
                           out.flush();
```

```
                            continue;
                        }

                        // Process the request and generate a response
                        String response = processRequest(message);


                        // Accumulate the JSON request message
                        requestMessages.append(request).append("\n");

                        // Serialize the response object to JSON
                        ResponseMessage responseMessage = new
ResponseMessage(response, blockchain.getChainSize());
                        String jsonResponse = responseMessage.toJSON();

                        // Accumulate the JSON response message
                        responseMessages.append(jsonResponse).append("\n");

                        // Send the JSON response to the client
                        out.println(jsonResponse);
                        out.flush();

                    } catch (JsonSyntaxException e) {
                        // Handle JSON parsing error
                        System.out.println("Invalid JSON request: " +
e.getMessage());
                        // Optionally, you can send an error response to the
client
                        out.println("Invalid JSON request: " + e.getMessage());
                        out.flush();
                    } catch (Exception e) {
                        throw new RuntimeException(e);
                    }
                }
            }
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        } finally {
            try {
                // Close server socket
                if (serverSocket != null) {
                    serverSocket.close();
                }
                // Close client socket
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    // Method to process the request and generate a response
    private static String processRequest(String request) {
        String[] parts = request.split(" ", 4);
        String clientID = parts[0];
        int choice = Integer.parseInt(parts[1]);
        StringBuilder response = new StringBuilder();
        response.append("ClientID = ").append(clientID).append("\n");
        switch (choice) {
            case 0:
                response.append(getBasicBlockchainStatus());
                break;
            case 1:
                response.append(addTransaction(parts[3],
Integer.parseInt(parts[2])));
                break;
            case 2:
                response.append(verifyBlockchain());
                break;
            case 3:
                response.append(viewBlockchain());
                break;
            case 4:
                response.append(corruptChain(Integer.parseInt(parts[2]),
parts[3]));
                break;
            case 5:
                response.append(repairChain());
                break;
            case 6:
                response.append("Exiting...").append("\n");
                break;
            default:
                response.append("Invalid choice").append("\n");
                break;
        }
        responseMessages.append(response);
        return response.toString();
    }

    // Method to verify if the provided public key hashes to the client ID
    private static boolean verifyClientID(String publicKey,BigInteger modulus,
String clientID) {
        String pubKey = publicKey +modulus.toString();
        System.out.println("client" + clientID);
        System.out.println(pubKey);
```

```java
        // Hash the public key string
        String hashedPublicKey = String.valueOf(computeHash(pubKey));
        System.out.println("hashed: " + hashedPublicKey);

        // Check if the hashed public key string matches the client ID string
        return hashedPublicKey.equals(clientID);
    }

    public static boolean verifySignature(String messageToCheck, String sig,
String e, BigInteger n)throws Exception  {

        // Take the encrypted string and make it a big integer
        BigInteger encryptedHash = new BigInteger(sig);
        // Decrypt it
        BigInteger r = new BigInteger(e);
        BigInteger decryptedHash = encryptedHash.modPow(r, n);

        // Get the bytes from messageToCheck
        byte[] bytesOfMessageToCheck = messageToCheck.getBytes("UTF-8");

        // compute the digest of the message with SHA-256
        MessageDigest md = MessageDigest.getInstance("SHA-256");

        byte[] messageToCheckDigest = md.digest(bytesOfMessageToCheck);

        // messageToCheckDigest is a full SHA-256 digest
        // take two bytes from SHA-256 and add a zero byte
        byte[] extraByte = new byte[3];
        extraByte[0] = 0;
        extraByte[1] = messageToCheckDigest[0];
        extraByte[2] = messageToCheckDigest[1];

        // Make it a big int
        BigInteger bigIntegerToCheck = new BigInteger(extraByte);

        // inform the client on how the two compare
        if(bigIntegerToCheck.compareTo(decryptedHash) == 0) {

            return true;
        }
        else {
            return false;
        }
    }


    private static BigInteger computeHash(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(input.getBytes());
            return new BigInteger(1, hash);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return BigInteger.ZERO;
        }
    }

    // Method to get basic blockchain status
    private static String getBasicBlockchainStatus() {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        response.append("Current size of chain:
").append(blockchain.getChainSize()).append("\n");
        response.append("Difficulty of most recent block:
").append(blockchain.getLatestBlock().getDifficulty()).append("\n");
        response.append("Total difficulty for all blocks:
").append(blockchain.getTotalDifficulty()).append("\n");
        try {
            // Attempt to compute hashes per second
            BlockChain.computeHashesPerSecond();
            response.append("Approximate hashes per second on this machine:
").append(blockchain.getHashesPerSecond()).append("\n");
        } catch (NoSuchAlgorithmException e) {
            // If an exception occurs, include error message in the response
            response.append("Failed to compute hashes per second:
").append(e.getMessage()).append("\n");
        }
        response.append("Expected total hashes required for the whole chain:
").append(String.format("%.6f",
blockchain.getTotalExpectedHashes())).append("\n");
        response.append("Nonce for most recent block:
").append(blockchain.getLatestBlock().getNonce()).append("\n");
        response.append("Chain hash:
").append(blockchain.getChainHash()).append("\n");
        return response.toString();
    }

    // Method to add a transaction to the blockchain
    private static String addTransaction(String data, int difficulty) {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        String newData = data;
        Block newBlock = new Block(blockchain.getChainSize(),
blockchain.getTime(), newData, difficulty);
        long startTime = System.currentTimeMillis();
        blockchain.addBlock(newBlock);
        long endTime = System.currentTimeMillis();
```

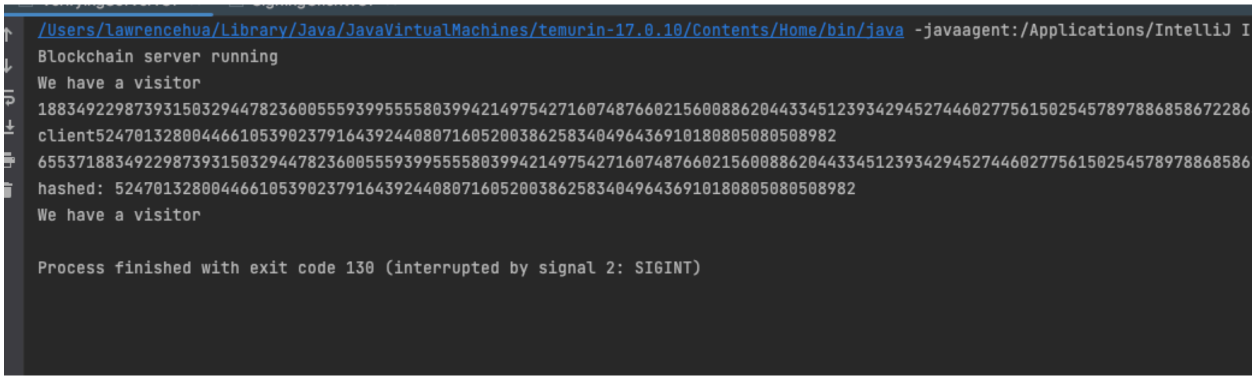
```java
        response.append("Total execution time to add this block was
").append(endTime - startTime).append(" milliseconds.").append("\n");
        return response.toString();
    }

    // Method to verify the blockchain
    private static String verifyBlockchain() {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        response.append("Verifying entire chain\n");
        long startTime = System.currentTimeMillis();
        String validationResult = blockchain.isChainValid();
        long endTime = System.currentTimeMillis();
        response.append("Chain verification:
").append(validationResult).append("\n");
        response.append("Total execution time required to verify the chain was
").append(endTime - startTime).append(" milliseconds").append("\n");
        return response.toString();
    }

    // Method to view the blockchain
    private static String viewBlockchain() {
        return "View the Blockchain\n" + blockchain.toString();
    }

    // Method to corrupt the blockchain
    private static String corruptChain(int blockIndex, String newData) {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        blockchain.chain.get(blockIndex).setData(newData);
        response.append("Block ").append(blockIndex).append(" now holds
").append(newData).append("\n");
        return response.toString();
    }

    // Method to repair the blockchain
    private static String repairChain() {
        // StringBuilder to accumulate response
        StringBuilder response = new StringBuilder();
        response.append("Repairing the entire chain\n");
        long startTime = System.currentTimeMillis();
        blockchain.repairChain();
        long endTime = System.currentTimeMillis();
        response.append("Total execution time required to repair the chain was
").append(endTime - startTime).append(" milliseconds.").append("\n");
        return response.toString();
    }
    private static byte[] computeSHA256Hash(String data) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            return digest.digest(data.getBytes());
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }

}
```

Project2Task2ClientConsole



```
/Users/lawrencehua/Library/Java/JavaVirtualMachines/temurin-17.0.10/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_r
id stromg65537188349229873931503294478236005559399555580399421497542716074876602156008862044334512393429452744602775615025457897886858672286361668967552 9
52470132800446610539023791643924408071605200386258340496436910180805080508982
RSA Public Key (e,n): 65537,18834922987393150329447823600555939955558039942149754271607487660215600886204433451239342945274460277561502545789788685867228
RSA Private Key (d,n): 6143903193989513827958795381733751687900266687264864836938292433283322638284913534653625791294649004588397415565163228800261506477
Client ID: 52470132800446610539023791643924408071605200386258340496436910180805080508982
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 recomputing hashes.
6. Exit.
Enter your choice:
```

Project2Task2ServerConsole



```
/Users/lawrencehua/Library/Java/JavaVirtualMachines/temurin-17.0.10/Contents/Home/bin/java -javaagent:/Applications/IntelliJ I
Blockchain server running
We have a visitor
188349229873931503294478236005559399555580399421497542716074876602156008862044334512393429452744602775615025457897886858672286
client52470132800446610539023791643924408071605200386258340496436910180805080508982
655371883492298739315032944782360055593995555803994214975427160748766021560088620443345123934294527446027756150254578978868586
hashed: 52470132800446610539023791643924408071605200386258340496436910180805080508982
We have a visitor

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

I was almost complete with task 2. I just couldnt properly verify the signature. Im able to represent multiple clients, but as you can see it doesnt display properly due to the verify signature check. But, if there was no verification it works properly.

I used ChatGPT in all tasks to help generate the code!
For task 2, I used the examples provided in the doc in order to satisfy the requirements as best as i could. Thank you very much for your consideration!

🙂