Praveen Ramesh - pramesh2@andrew.cmu.edu

Task 0 Execution

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 2386000
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 313
Chain hash:
008FAD82D83741A6DB80BC0F075F886C3AD0917FD12C6B40225100E8FE26C32B
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Alice pays Bill 100 DSCoin

Total execution time to add this block was 2 milliseconds
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Bill pays Clara 50 DSCoin
Total execution time to add this block was 2 milliseconds
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Clara pays Daisy 10 DSCoin
Total execution time to add this block 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.
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
{"ds_chain":[{"index":0,"timestamp":"2023-03-17
02:28:19.313","data":"Genesis","previousHash":"","nonce":313,"difficulty":2},{"index":1,"timestam
p":"2023-03-17 02:28:52.350","data":"Alice pays Bill 100
DSCoin","previousHash":"008FAD82D83741A6DB80BC0F075F886C3AD0917FD12C6B40225
100E8FE26C32B","nonce":19,"difficulty":2},{"index":2,"timestamp":"2023-03-17
02:29:14.712","data":"Bill pays Clara 50
DSCoin","previousHash":"00ECBAF149F89EF175AF44CB5E7556A5D3687B6EC1D996B4C87
313EF0D51E4E2","nonce":104,"difficulty":2},{"index":3,"timestamp":"2023-03-17
02:29:32.952","data":"Clara pays Daisy 10
DSCoin","previousHash":"00BCCEE9EEC674350B0894AA32CA85C016E27AA33158AF07897
B082D29CA26D4","nonce":16,"difficulty":2}],"chainHash":"008B25DFB69024C73362E421E66B
C8065F2D527832095CC29FC130AF78C7E6DD"}
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
4
corrupt the Blockchain
Enter block ID of block to corrupt
1
Enter new data for block 1
Alice pays Bill 76 DSCoin
Block 1 now holds Alice pays Bill 76 DSCoin
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
3
{"ds_chain":[{"index":0,"timestamp":"2023-03-17
02:28:19.313","data":"Genesis","previousHash":"","nonce":313,"difficulty":2},{"index":1,"timestam
p":"2023-03-17 02:28:52.350","data":"Alice pays Bill 76
DSCoin","previousHash":"008FAD82D83741A6DB80BC0F075F886C3AD0917FD12C6B40225
100E8FE26C32B","nonce":19,"difficulty":2},{"index":2,"timestamp":"2023-03-17
02:29:14.712","data":"Bill pays Clara 50
DSCoin","previousHash":"00ECBAF149F89EF175AF44CB5E7556A5D3687B6EC1D996B4C87
313EF0D51E4E2","nonce":104,"difficulty":2},{"index":3,"timestamp":"2023-03-17
02:29:32.952","data":"Clara pays Daisy 10
DSCoin","previousHash":"00BCCEE9EEC674350B0894AA32CA85C016E27AA33158AF07897
B082D29CA26D4","nonce":16,"difficulty":2}],"chainHash":"008B25DFB69024C73362E421E66B
C8065F2D527832095CC29FC130AF78C7E6DD"}
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 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
Chain Verification : TRUE
Total execution time to verify the chain was 1 milliseconds
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
1
Enter difficulty > 0
4
Enter transaction
Daisy pays Sean 25 DSCoin
Total execution time to add this block was 133 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: 2386000
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 108670
Chain hash:
00006CB537F87EBDD0BAE48228A281F57BC03583005491B702E596B6B71E397C
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
6

Process finished with exit code 0

Task 0 Block.java

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2@andrew.cmu.edu
The Block class represents a block in a blockchain. It contains data
fields for the index,timestamp, data, previous hash,nonce, and difficulty
of the block. It also has methods for calculating the hash of the block using
the SHA-256 algorithm, performing a proof of work to find a hash with a
specified
number of leading zeros, and converting the block to a JSON string using
the Gson library.
*/
//import required packages
package blockchaintask0;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
//A public class that has all the required methods
public class Block extends java.lang.Object{
   //index representing the idex of the block
   int index;
   //timestamp to store the time of creation
   java.sql.Timestamp timestamp;
   //data to store the transaction
   java.lang.String data;
   //previoushash to store the hashvalue of the previous block
   String previousHash;
   //BigInteger value determined by a proof of work routine
   BigInteger nonce;
   //an int that specifies the minimum number of left most hex digits needed
by a proper hash
   int difficulty;
   /**
    Constructs a new Block object with the specified index, timestamp,
    data, and difficulty.
    @param index the index of the block in the blockchain
    @param timestamp the timestamp indicating when the block was created
    @param data the data stored in the block
    @param difficulty the difficulty level for the proof of work algorithm
    */
   public Block(int index, Timestamp timestamp, String data, int difficulty) {
       this.index = index;
       this.timestamp = timestamp;
       this.data = data;
       this.difficulty = difficulty;
```

```
}
//getter method for the index
public int getIndex() {
    return index;
}
//getter method for the timestamp
public Timestamp getTimestamp() {
    return timestamp;
}
//getter method for the data
public String getData() {
    return data;
}
//getter method for the previousHash
public String getPreviousHash() {
    return previousHash;
}
//getter method for the nonce
public BigInteger getNonce() {
    return nonce;
}
//getter method for the difficulty
public int getDifficulty() {
    return difficulty;
}
//setter method for index
public void setIndex(int index) {
    this.index = index;
}
//setter method for the timestamp
public void setTimestamp(Timestamp timestamp) {
    this.timestamp = timestamp;
}
//setter method for data
public void setData(String data) {
    this.data = data;
}
//setter method for previousHash
public void setPreviousHash(String previousHash) {
    this.previousHash = previousHash;
}
//setter method for difficulty
public void setDifficulty(int difficulty) {
    this.difficulty = difficulty;
}
//method for calculating hash of the block using SHA=256
public java.lang.String calculateHash(){
    //declare MessageDigest object for hashing
    MessageDigest md = null;
```

```
        //declare hash string
        String hash;
        //Initialize the message to hash
        String message = index + timestamp.toString() + data + previousHash +
nonce.toString() + String.valueOf(difficulty);
        try {
            //get the SHA-256 instance
            md = MessageDigest.getInstance("SHA-256");
            //update the digest
            md.update(message.getBytes());
            //completes the hash computation
            hash = bytesToHex(md.digest());
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
        //return the hash String
        return hash;
    }

    /**
     * Method that carries out a proof-of-work calculation to locate a
     * hash value that satisfies the criterion for difficulty defined for this
block.
     * @return the hash which is computed that meets the difficulty
     */
    public java.lang.String proofOfWork() {
        //Initialze the nonce to 0
        nonce = BigInteger.valueOf(0);
        //Create a check string with required difficulty to compare it with the
hash value
        String check = "0".repeat(difficulty);
        //call the calculateHash() to compute the hash
        String hash = calculateHash();
        //while loop to get the hash that matches the required difficulty
        while (!hash.substring(0, difficulty).equals(check)) {
            //adding one to the nonce
            nonce = nonce.add(BigInteger.valueOf(1));
            //computing hash
            hash = calculateHash();
        }
        //return the computed hash that matches the
        return hash;
    }

    // Code from stack overflow
    //
https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-h
ex-string-in-java
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();
```

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

    /**
     * toString method overridden to parse the object to JSON string
     * @return the JSON object of the block
     */
    public String toString(){
        Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd
HH:mm:ss.SSS").create();
        // Serialize to JSON
        return gson.toJson(this);

    }
}
```

Task 0 BlockChain.java

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2@andrew.cmu.edu
The BlockChain class represents a blockchain of all the transactions. It
contains data
fields for the blocks, chainHash and hashesPerSecond of the chain. It also has
methods for
adding a block,verifying the blockchain, viewing the blockchain, corrupting
and repairing the
blockchain.
*/
//import required packages
package blockchaintask0;
import com.google.gson.Gson;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
//A public class that has all the required methods
public class BlockChain extends java.lang.Object{
   //An arraylist which contains all the blocks
   private ArrayList<Block> blocks;
   //chainHash which contains the hash value of the last block in the chain
   private String chainHash;
   //hashes computed per second by the machine
   private int hashesPerSecond;
   //Constructor to initialize the members of the class
   public BlockChain(){
       blocks=new ArrayList<Block>();
       chainHash ="";
       hashesPerSecond=0;
   }

   /**
    * Method that adds a block to the blockchain after performing the
    * proof of work. ChainHash is assigned from the last block's hash value
    * @param newBlock
    */
   public void addBlock(blockchaintask0.Block newBlock){
       //if the size of the block is non-zero then set the previousHash of the
```

```
        //new block to the chain's current chainHash, else the previousHash
will
        //be "" for chain containing just one block
        if(blocks.size()!=0){
            newBlock.setPreviousHash(chainHash);
        }
        //perform the proof of work for the newly added block
        newBlock.proofOfWork();
        //add the block to the arraylist
        blocks.add(newBlock);
        //set the chainHash as the new block's hash value
        chainHash=newBlock.calculateHash();
    }

    /**
     * Method to check whether a chain is valid by comparing the block's hash
     * with the number of leftmost 0's (proof of work) as specified in the
difficulty field
     * and comparing the hash with the previousHash of the next block. The
corresponding
     * chainHash of the chain is also checked.
     * @return TRUE if the chain is valid, else returns FALSE with an
appropriate error message
     */
    public java.lang.String isChainValid(){
        //checks the size of the block
        if(blocks.size()==1){
            //get the first genesis block
            Block b=blocks.get(0);
            //compute the hash of the block
            String hash= b.calculateHash();
            //Initialize a string to check the proof of work
            String check = "0".repeat(b.getDifficulty());
            //compares the hash with the leftmost 0's
            if(!hash.substring(0, b.getDifficulty()).equals(check)){
                //if comparison fails then return false and the error message
                return "FALSE \nImproper hash on node 0 Does not begin with 00";
            } else if (!hash.equals(chainHash)) {
                //if the chainHash doesn't match with the computed hash then
                //return false with the error message
                return "FALSE \nHash value and chainHash are not matching";
            }
            //else return true
            else {
                return "TRUE";
            }
        }
        //logic for chain containing more than one block
        else {
```

```
            //loop for iterating over the chain
            for (int i=0;i<blocks.size();i++){
                //get the ith block
                Block b = blocks.get(i);
                //compute the hash of the block
                String hash = b.calculateHash();
                //check if the block is the last block in the chain
                if(i!=blocks.size()-1) {
                    //Initialize the check string to compare hash with the
difficulty
                    String check = "0".repeat(b.getDifficulty());
                    //check the substring of the hash
                    if (!hash.substring(0, b.getDifficulty()).equals(check)) {
                        //if the comparison fails then return false with the
error message
                        return "FALSE \nImproper hash on node "+ i +" Does not
begin with 00";
                    }
                    //checks the block's hash value with the next block's
previousHash value
                    if (!hash.equals(blocks.get(i + 1).previousHash)){
                        //if the comparison fails then return FALSE with an
error message
                        return "FALSE \nHash value of node "+ i +" is not
matching with node "+(i+1)+" previosHash";
                    }
                }
                //check condition for the last block in the chain
                else {
                    //check whether the hash value matches the chainHash of the
blockchain
                    if (!hash.equals(chainHash)){
                        //if the condition fails then return FALSE with the
corresponding error
                        return "FALSE \nHash value and chainHash are not
matching";
                    }
                }
            }
            //return TRUE if all the check cases are passed
            return "TRUE";
        }
    }

    /**
     * Method that 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.It computes new proof
```

```
     * of work based on the difficulty specified in the Block
     */
    public void repairChain()
    {   //loop for iterating over the chain
        for (int i=0;i<blocks.size();i++)
        {   //compute the hash of the ith block
            String hash = blocks.get(i).calculateHash();
            //check if the block is the last block in the chain
            if(i!=blocks.size()-1)
            {   //checks the block's hash value with the next block's
previousHash value
                if (!hash.equals(blocks.get(i + 1).previousHash))
                {   //if the comparison fails then compute the proof of work
again
                    blocks.get(i).proofOfWork();
                    //set the previousHash of the next block to the new hash

blocks.get(i+1).setPreviousHash(blocks.get(i).calculateHash());
                }
            }
            //check condition for the last block in the chain
            else
            {   //check whether the hash value matches the chainHash of the
blockchain
                if (!hash.equals(chainHash)){
                    //if the comparison fails then compute the proof of work
again
                    blocks.get(i).proofOfWork();
                    //set the previousHash of the next block to the new hash
                    chainHash=blocks.get(i).calculateHash();
            }
            }
        }
    }
    //getter method to get the chainHash
    public java.lang.String getChainHash(){
        return chainHash;
    }

    //getter method to get the latestBlock in the blockChain
    public blockchaintask0.Block getLatestBlock(){
        return blocks.get(blocks.size()-1);
    }
    //getter method to get the chain size
    public int getChainSize(){
        return blocks.size();
    }

    /**
```

```
    * This method computes exactly 2 million hashes and times how long that
process takes
    * and computes the hashes per second from that value
    */
   public void computeHashesPerSecond(){
       //Initialize the message to hash
       String messageToHash="00000000";
       String hash;
       MessageDigest md = null;
       //get the start time
       long start = System.currentTimeMillis();
       //loop for hashing 2 million times
       for (int i=0;i<2000000;i++){
           try {
               //get the SHA-256 instance
               md = MessageDigest.getInstance("SHA-256");
               //compute the hash
               md.update(messageToHash.getBytes());
               //convert it to hex string
               hash = bytesToHex(md.digest());
           } catch (NoSuchAlgorithmException e) {
               throw new RuntimeException(e);
           }
       }
       //get the stop time
       long stop = System.currentTimeMillis();
       //compute hashesPerSecond
       hashesPerSecond= (int) (2000000/(stop-start))*1000;
   }

   //getter method to get the hashesPerSecond
   public int getHashesPerSecond(){
       return hashesPerSecond;
   }

   //getter  method to get the block
   public blockchaintask0.Block getBlock(int i){
       return blocks.get(i);
   }

   //getter method to get the total difficulty of the chain
   public int getTotalDifficulty(){
       //Initialze the totalDifficulty to 0
       int totalDifficulty=0;
       //loop over each block
       for (Block b:blocks) {
           //sum the difficulty
           totalDifficulty=totalDifficulty+b.getDifficulty();
       }
```

```
        //return the totalDifficulty
        return totalDifficulty;
    }

    //method to get the totalExpectedHashes
    public double getTotalExpectedHashes(){
        //Initialze the totalExpectedHashes to 0
        double totalExpectedHashed=0;
        //loop over each block
        for (Block b: blocks) {
            //compute the total expected hashed
            totalExpectedHashed=
(totalExpectedHashed+Math.pow(16,b.getDifficulty()));
        }
        //return the totalExpectedHashes
        return totalExpectedHashed;
    }

    //method to get the difficulty of most recent block
    public int difficultyMostRecent(){
        return blocks.get(blocks.size()-1).getDifficulty();
    }

    //method to get the nonce of most recent block
    public BigInteger nonceMostRecent(){
        return blocks.get(blocks.size()-1).getNonce();
    }

    /**
     * Method that overrides the toString() to get a JSON format of the
     * blockchain and the chain's chainHash
     * @return the JSON format of the blockchain
     */
    public java.lang.String toString(){
        //Declare and initialize the GSON's JSON object
        JsonObject jsonObject = new JsonObject();
        //Declare and initialize the GSON's JSON array
        JsonArray jsonArray = new JsonArray();
        //Create a GsonBuilder
        Gson gson = new GsonBuilder().create();
        //loop over all blocks
        for (Block b :blocks)
        {   //get the toString() of each block and convert it to JSON object
            JsonObject json = gson.fromJson(b.toString(), JsonObject.class);
            //add the JSON object to the JSON array
            jsonArray.add(json);
        }
        //append the JSON elements to a final JSONObject
        jsonObject.add("ds_chain",jsonArray);
```

```
        jsonObject.addProperty("chainHash",this.chainHash);

        //return the String format of the JSONObject
        return jsonObject.toString();
    }

    public static java.sql.Timestamp getTime(){
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss.SSS");
        LocalDateTime currentDateTime = LocalDateTime.now();
        Timestamp formattedDateTime = Timestamp.valueOf(new
String(currentDateTime.format(formatter)));
        return formattedDateTime;
    }


    // Code from stack overflow
    //
https://stackoverflow.com/questions/9655181/how-to-convert-a-byte-array-to-a-h
ex-string-in-java
    private static final char[] HEX_ARRAY = "0123456789ABCDEF".toCharArray();
    /**
     * Method to convert bytes array to hex String
     * @param bytes
     * @return the hex String format of the input bytes array
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
        }
        return new String(hexChars);
    }

    /**
     * Main method that acts as a test driver for your Blockchain. It will
begin by
     * creating a BlockChain object and then adding the Genesis block to the
chain.
     * The Genesis block will be created with an empty string as the pervious
hash and a difficulty of 2.
     * All other methods are called based on the user's choice
     * @param args
     */
    public static void main(java.lang.String[] args){
        //Initialize the index to 0
        int index=0;
```

```
        //Create scanner object
        Scanner scanner = new Scanner(System.in);
        //Initialize the blockChain
        BlockChain blockChain=new BlockChain();
        //Call the computeHashesPerSecond method
        blockChain.computeHashesPerSecond();
        //Create the genesis block
        Block genesis=new Block(0,getTime(),"Genesis",2);
        //add 1 to the index
        index=index+1;
        //set the previousHash of the genesis block to "" string
        genesis.setPreviousHash("");
        //add the genesis block to the chain
        blockChain.addBlock(genesis);
        //declare the choice variable
        int choice;
        //create an infinite loop
        while(true){
             /*
            Time Data Analysis:
            For addBlock():
            When the difficulty is 2 - 5 milliseconds
            When the difficulty is 4 - 50 milliseconds
            When the difficulty is 6 - 42357 milliseconds

            For isChainValid():
            The time for execution doesn't change

            For repairChain():
            When the difficulty is 6 - 44406 milliseconds

            Therefore, when the difficulty increases the time to find the proof
of work increases drastically while adding
            the new block.Thus, when difficulty increases time taken to add the
block also increases drastically.
            The isChainValid() method doesn't take that much time because it
doesn't do proof of work.
            However, repairChain() takes a lot of time when difficulty
increases as it has to do the proof of block again,
            which takes a lot of time when difficulty is higher
             */
            //display the menu
            System.out.println("0. View basic blockchain status.\n" +
                    "\n" +
                    "1. Add a transaction to the blockchain.\n" +
                    "\n" +
                    "2. Verify the blockchain.\n" +
                    "\n" +
                    "3. View the blockchain.\n" +
```

```
                    "\n" +
                    "4. Corrupt the chain.\n" +
                    "\n" +
                    "5. Hide the corruption by repairing the chain.\n" +
                    "\n" +
                    "6. Exit.");
            //get the choice
            choice=Integer.parseInt(scanner.nextLine());
            //check the choice
            if(choice==6){
                System.exit(0);
            }
            //if the choice is 0 display all the details of the blockchain
            if(choice==0){
                //display the details by calling the required methods
                System.out.println("Current size of chain:
"+blockChain.getChainSize());
                System.out.println("Difficulty of most recent block: "+
blockChain.difficultyMostRecent());
                System.out.println("Total difficulty for all blocks:
"+blockChain.getTotalDifficulty());
                System.out.println("Approximate hashes per second on this
machine: "+blockChain.getHashesPerSecond());
                System.out.println("Expected total hashes required for the whole
chain: "+blockChain.getTotalExpectedHashes());
                System.out.println("Nonce for most recent block:
"+blockChain.nonceMostRecent());
                System.out.println("Chain hash: "+blockChain.chainHash);
            }
            //Adding the block
            if(choice==1){
                //declare difficulty
                int difficulty;
                //declare transaction
                String transaction;
                //get the difficulty and transaction
                System.out.println("Enter difficulty > 0");
                difficulty=Integer.parseInt(scanner.nextLine());
                System.out.println("Enter transaction");
                transaction=scanner.nextLine();
                //get the start time
                long start = System.currentTimeMillis();
                //create a new block
                Block block= new Block(index,getTime(),transaction,difficulty);
                //add 1 to the index
                index=index+1;
                //call the addBlock method to add the new block
                blockChain.addBlock(block);
                //get the stop time
```

```
                long stop = System.currentTimeMillis();
                //display the execution time
                System.out.println("Total execution time to add this block was
"+(stop-start)+" milliseconds");

            }
            //Chain verification
            if(choice==2){
                //get the start time
                long start = System.currentTimeMillis();
                //call the isChainValid method to verify the chain
                System.out.println("Chain Verification : "+
blockChain.isChainValid());
                //get the stop time
                long stop = System.currentTimeMillis();
                //display the execution time
                System.out.println("Total execution time to verify the chain was
"+(stop-start)+" milliseconds");
            }
            //display the blockchain
            if(choice==3){
                //call the toString method to display the JSON
                System.out.println(blockChain.toString());
            }
            //Corrupt the blockchain
            if(choice==4){
                System.out.println("corrupt the Blockchain");
                //get the ID to corrupt
                System.out.println("Enter block ID of block to corrupt");
                int ID=Integer.parseInt(scanner.nextLine());
                //get the new transaction from the user
                System.out.println("Enter new data for block "+ID);
                String newData=scanner.nextLine();
                //set the new transaction to the block through the setter
method
                blockChain.blocks.get(ID).setData(newData);
                //display the result
                System.out.println("Block "+ID+" now holds "+newData);
            }
            //Repair the chain
            if(choice==5){
                //get the start time
                long start = System.currentTimeMillis();
                //call the  method to repair the blackchain
                blockChain.repairChain();
                //get the stop time
                long stop = System.currentTimeMillis();
                //display the execution time
```

```
                System.out.println("Total execution time to repair the chain was
"+(stop-start)+" milliseconds");
            }
        }}}
```

Task 1 Client Side Execution

The BlockChain client is running
Enter the BlockChainServer port number:6789
The BlockChain 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: 2328000
Expected total hashes required for the whole chain: 256
Nonce for most recent block: 486
Chain hash:
00EECA21AA1D6EDD90559EB934B2C773E97CFE219EF4BF732A084CF4E8C77EA2
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.

1
Enter difficulty > 0
2
Enter transaction
Alice pays Bill 100 DSCoin
Total execution time to add this block was 3 milliseconds
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
1
Enter difficulty > 0
2
Enter transaction
Bill pays Clara 50 DSCoin
Total execution time to add this block was 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
2
Enter transaction
Clara pays Daisy 10 DSCoin
Total execution time to add this block was 2 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
2
Chain Verification : TRUE
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
3
View the Blockchain
{"ds_chain":[{"index":0,"timestamp":"2023-03-17
02:51:53.650","data":"Genesis","previousHash":"","nonce":486,"difficulty":2},{"index":1,"timestam
p":"2023-03-17 02:52:26.487","data":"Alice pays Bill 100
DSCoin","previousHash":"00EECA21AA1D6EDD90559EB934B2C773E97CFE219EF4BF732A
084CF4E8C77EA2","nonce":145,"difficulty":2},{"index":2,"timestamp":"2023-03-17
02:52:43.450","data":"Bill pays Clara 50
DSCoin","previousHash":"00C8A080236DCDBFA79A1917C9749375E42D20ED3261E69AB55
1C5E1DD94A1BD","nonce":109,"difficulty":2},{"index":3,"timestamp":"2023-03-17
02:52:58.303","data":"Clara pays Daisy 10
DSCoin","previousHash":"00F4A1583AE79CCA05292E9FB4C2B45B8DA1E485546A79126C2
6C54EE58DC79E","nonce":488,"difficulty":2}],"chainHash":"0069FEEF7CEAD6F22340B9AC99
AA77220126E34951F9DC67F895E2876C9A623F"}
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
4
corrupt the Blockchain
Enter block ID of block to corrupt
1
Enter new data for block 1
Alice pays Bill 76 DSCoin
Block 1 now holds Alice pays Bill 76 DSCoin
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
3
View the Blockchain
{"ds_chain":[{"index":0,"timestamp":"2023-03-17
02:51:53.650","data":"Genesis","previousHash":"","nonce":486,"difficulty":2},{"index":1,"timestam
p":"2023-03-17 02:52:26.487","data":"Alice pays Bill 76
DSCoin","previousHash":"00EECA21AA1D6EDD90559EB934B2C773E97CFE219EF4BF732A
084CF4E8C77EA2","nonce":145,"difficulty":2},{"index":2,"timestamp":"2023-03-17
02:52:43.450","data":"Bill pays Clara 50
DSCoin","previousHash":"00C8A080236DCDBFA79A1917C9749375E42D20ED3261E69AB55
1C5E1DD94A1BD","nonce":109,"difficulty":2},{"index":3,"timestamp":"2023-03-17
02:52:58.303","data":"Clara pays Daisy 10
DSCoin","previousHash":"00F4A1583AE79CCA05292E9FB4C2B45B8DA1E485546A79126C2

6C54EE58DC79E","nonce":488,"difficulty":2}],"chainHash":"0069FEEF7CEAD6F22340B9AC99
AA77220126E34951F9DC67F895E2876C9A623F"}
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 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.
5
Total execution time to repair 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.
2
Chain Verification : TRUE
Total execution time to verify the chain was 0 milliseconds
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
1
Enter difficulty > 0
4
Enter transaction
Daisy pays Sean 25 DSCoin
Total execution time to add this block was 64 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: 2328000
Expected total hashes required for the whole chain: 66560

Nonce for most recent block: 38536
Chain hash:
0000CF254DB78AD3C0EE2E0A8B4987E5B96079C077573D9F8DF3320346BA825C
0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit.
6

Process finished with exit code 0

Task 1 Server Side Execution

Enter the port number for the server to listen:6789
Blockchain server running
We have a visitor
Response
:{"selection":"0","size":1,"chainHash":"00EECA21AA1D6EDD90559EB934B2C773E97CFE219E
F4BF732A084CF4E8C77EA2","totalHashes":256,"totalDiff":2,"recentNonce":486,"diff":2,"hps":2
328000}
Adding a block
Setting response to Total execution time to add this block was 3 milliseconds
{"selection":"1","response":"Total execution time to add this block was 3 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 1 milliseconds
{"selection":"1","response":"Total execution time to add this block was 1 milliseconds"}
Adding a block
Setting response to Total execution time to add this block was 2 milliseconds
{"selection":"1","response":"Total execution time to add this block was 2 milliseconds"}
Verifying entire chain
Chain Verification : TRUE
Total execution time to verify the chain was 0 milliseconds
Setting response to Total execution time to verify the chain was 0 milliseconds
View the Blockchain
Setting response to {"ds_chain":[{"index":0,"timestamp":"2023-03-17
02:51:53.650","data":"Genesis","previousHash":"","nonce":486,"difficulty":2},{"index":1,"timestam
p":"2023-03-17 02:52:26.487","data":"Alice pays Bill 100
DSCoin","previousHash":"00EECA21AA1D6EDD90559EB934B2C773E97CFE219EF4BF732A
084CF4E8C77EA2","nonce":145,"difficulty":2},{"index":2,"timestamp":"2023-03-17
02:52:43.450","data":"Bill pays Clara 50
DSCoin","previousHash":"00C8A080236DCDBFA79A1917C9749375E42D20ED3261E69AB55
1C5E1DD94A1BD","nonce":109,"difficulty":2},{"index":3,"timestamp":"2023-03-17
02:52:58.303","data":"Clara pays Daisy 10
DSCoin","previousHash":"00F4A1583AE79CCA05292E9FB4C2B45B8DA1E485546A79126C2
6C54EE58DC79E","nonce":488,"difficulty":2}],"chainHash":"0069FEEF7CEAD6F22340B9AC99
AA77220126E34951F9DC67F895E2876C9A623F"}
corrupt the Blockchain
Block 1 now holds Alice pays Bill 76 DSCoin
Setting response to Block 1 now holds Alice pays Bill 76 DSCoin
View the Blockchain

Setting response to {"ds_chain":[{"index":0,"timestamp":"2023-03-17 02:51:53.650","data":"Genesis","previousHash":"","nonce":486,"difficulty":2},{"index":1,"timestamp":"2023-03-17 02:52:26.487","data":"Alice pays Bill 76 DSCoin","previousHash":"00EECA21AA1D6EDD90559EB934B2C773E97CFE219EF4BF732A084CF4E8C77EA2","nonce":145,"difficulty":2},{"index":2,"timestamp":"2023-03-17 02:52:43.450","data":"Bill pays Clara 50 DSCoin","previousHash":"00C8A080236DCDBFA79A1917C9749375E42D20ED3261E69AB551C5E1DD94A1BD","nonce":109,"difficulty":2},{"index":3,"timestamp":"2023-03-17 02:52:58.303","data":"Clara pays Daisy 10 DSCoin","previousHash":"00F4A1583AE79CCA05292E9FB4C2B45B8DA1E485546A79126C26C54EE58DC79E","nonce":488,"difficulty":2}],"chainHash":"0069FEEF7CEAD6F22340B9AC99AA77220126E34951F9DC67F895E2876C9A623F"}
Verifying entire chain
Chain Verification : FALSE
Improper hash on node 1 Does not begin with 00
Total execution time to verify the chain was 3 milliseconds
Setting response to Total execution time to verify the chain was 3 milliseconds
Repairing the entire chain
Setting response to Total execution time to repair the chain was 1 milliseconds
Verifying entire chain
Chain Verification : TRUE
Total execution time to verify the chain was 0 milliseconds
Setting response to Total execution time to verify the chain was 0 milliseconds
Adding a block
Setting response to Total execution time to add this block was 64 milliseconds
{"selection":"1","response":"Total execution time to add this block was 64 milliseconds"}
Response :{"selection":"0","size":5,"chainHash":"0000CF254DB78AD3C0EE2E0A8B4987E5B96079C077573D9F8DF3320346BA825C","totalHashes":66560,"totalDiff":12,"recentNonce":38536,"diff":4,"hps":2328000}

Task 1 Client Source Code

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2@andrew.cmu.edu
The BlockChainClient class is a client program that communicates with a
BlockChainServer
over a socket connection.Through this program, users can see the fundamental
blockchain
state, add a transaction, verify the blockchain, view the blockchain, corrupt
the chain,
and then cover up the corruption by repairing the chain.The program uses user
input to decide
what action to take before sending a request message via a socket connection
to the server.
The request is handled by the server, which then replies with a message that
the client receives
and sees on the console. The Gson library is utilized by the Java application
for JSON
serialization and deserialization.
*/
//import the required packages
package blockchaintask1;
import com.google.gson.Gson;
import java.io.*;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;
//Referred my project2Task4 code for TCP
public class BlockChainClient {
   //declare a scanner object
   static Scanner scanner ;
   //declare a socket
   static Socket clientSocket;

   /**
    * The BlockChainClient class main method is in charge of initializing the
client socket
    * and corresponding with the server.The action to be taken is determined
by user input,
    * and the socketCommunication() method is used to contact the server.
```

```
    * When the user enters 6 to end the program, the while loop in this method
continues to execute.
    * @param args
    */

   public static void main(String args[])
   {
       try {
           //declare a sort
           int serverPort;
           //create a scanner object
           scanner = new Scanner(System.in);
           System.out.println("The BlockChain client is running");
           System.out.print("Enter the BlockChainServer port number:");
           //get the server port
           serverPort = Integer.parseInt(scanner.nextLine());
           //Create a socket for connection
           clientSocket = new Socket("localhost", serverPort);
           System.out.println("The BlockChain client is running");
           //declare the choice
           int choice;
           //infinite loop till the user exists
           while (true) {
               //display the menu options
               System.out.println("0. View basic blockchain status.\n" +
                       "\n" +
                       "1. Add a transaction to the blockchain.\n" +
                       "\n" +
                       "2. Verify the blockchain.\n" +
                       "\n" +
                       "3. View the blockchain.\n" +
                       "\n" +
                       "4. Corrupt the chain.\n" +
                       "\n" +
                       "5. Hide the corruption by repairing the chain.\n" +
                       "\n" +
                       "6. Exit.");
               //get the choice from the user
               choice = Integer.parseInt(scanner.nextLine());
               //call the socketCommunication if the choice is not 6 to
perform the required operation
               if (choice!=6) {
                   socketCommunication(choice);
               }
               //exit if the choice is 6
               else
               {   //exit the program
                   System.exit(0);
               }
```

```
            }
        }
        //catch clause
        catch (UnknownHostException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * socketCommunication method is responsible for communicating with the
server
     * through the client socket.It takes an integer choice as an input, which
     * determines the action to be performed by the server.The method creates a
     * RequestMessage object based on the user input and sends it to the
server.
     * It then receives a JSON response from the server and calls the
displayResults()
     * method to display the response to the user.
     * @param choice An integer which determines the action to be performed by
the server.
     */
    public static void socketCommunication(int choice)
    {
        try {
            //create a requestMessage object with the choice
            RequestMessage requestMessage= createRequestMessage(choice);
            //create a BufferedReader object for reading the message
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            //create a PrintWriter object for writing the  message to server
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
            //write the requestMessage
            out.println(requestMessage.getRequestMessage());
            out.flush();
            //get the response JSON from the server
            String responseJSONFromServer = in.readLine();
            //call the displayResults method with the response JSON
            displayResults(responseJSONFromServer);

        } catch (IOException e) {
            throw new RuntimeException(e);
        }
     }

    /*
```

```
    *createRequestMessage method creates a RequestMessage object based on the
user's choice.
    *The user's choice determines the type of RequestMessage to create.
    * @param choice
    * @return RequestMessage object
    */
   public static RequestMessage  createRequestMessage(int choice)
   {
       //declare a requestMessage object
       RequestMessage requestMessage=null;
       if (choice==0){
           //create an object with the choice if the choice is 0
           requestMessage=new RequestMessage(String.valueOf(choice));
       }
       if (choice==1){
           //declare difficulty
           int enteredDifficulty;
           //declare transaction
           String transaction;
           //get the difficulty and transaction from the user
           System.out.println("Enter difficulty > 0");
           enteredDifficulty=Integer.parseInt(scanner.nextLine());
           System.out.println("Enter transaction");
           transaction=scanner.nextLine();
           //create a RequestMessage object using choice,entered difficulty
and transaction
           requestMessage=new
RequestMessage(String.valueOf(choice),enteredDifficulty,transaction);
       }
       //if the choice is 2 then create a RequestMessage object with the
choice
       if (choice==2){
           requestMessage=new RequestMessage(String.valueOf(choice));
       }
       //if the choice is 3 then create a RequestMessage object with the
choice
       if (choice==3){
           requestMessage=new RequestMessage(String.valueOf(choice));
       }
       if (choice==4) {
           System.out.println("corrupt the Blockchain");
           System.out.println("Enter block ID of block to corrupt");
           //get the corruptID from the user
           int CorruptID=Integer.parseInt(scanner.nextLine());
           //get the new transaction for the block
           System.out.println("Enter new data for block "+CorruptID);
           String newTransaction=scanner.nextLine();
           //create a RequestMessage object using choice, newTransaction and
CorruptID
```

```
            requestMessage=new
RequestMessage(String.valueOf(choice),newTransaction,CorruptID);
        }
        //if the choice is 3 then create a RequestMessage object with the
choice
        if(choice==5){
            requestMessage=new RequestMessage(String.valueOf(choice));
        }
        //return the requestMessage object
        return requestMessage;
    }

    /**
     * displayResults method displays the results received from the server.
     * @param responseJSONFromServer a JSON string containing the response
message from the server.
     */
    public static void displayResults(String responseJSONFromServer)
    {  //declare a selection variable
        int selection;
        //create a new Gson object
        Gson gson=new Gson();
        //convert the responseJSONFromServer string to ResponseMessage object
        ResponseMessage messageFromServer =
gson.fromJson(responseJSONFromServer,ResponseMessage.class);
        //get the selection from getter method
        selection= Integer.parseInt(messageFromServer.getSelection());
        if (selection==0){
            //if the se;ection is 0 then use the corresponding getters to
display the results
            System.out.println("Current size of chain:
"+messageFromServer.getSize());
            System.out.println("Difficulty of most recent block:
"+messageFromServer.getDiff());
            System.out.println("Total difficulty for all blocks: "+
messageFromServer.getTotalDiff());
            System.out.println("Approximate hashes per second on this machine:
"+ messageFromServer.getHps());
            System.out.println("Expected total hashes required for the whole
chain: "+ messageFromServer.getTotalHashes());
            System.out.println("Nonce for most recent block: "+
messageFromServer.getRecentNonce());
            System.out.println("Chain hash:
"+messageFromServer.getChainHash());
        }

        if (selection==1){
            //directly print the response message from the server
            System.out.println(messageFromServer.getResponse());
```

```
        }
        if(selection==2){
            //directly print the response message from the server
            System.out.println(messageFromServer.getResponse());
        }
        if(selection==3){
            //directly print the response message from the server
            System.out.println("View the Blockchain");
            System.out.println(messageFromServer.getResponse());
        }
        if(selection==4){
            //directly print the response message from the server
            System.out.println(messageFromServer.getResponse());
        }
        if(selection==5){
            //directly print the response message from the server
            System.out.println(messageFromServer.getResponse());
        }
    }
}
```

RequestMessage class:

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2@andrew.cmu.edu
The request message that the client sends to the blockchain server is
represented by this class.
It includes details on the kind of request being performed.
The class comprises three constructors, each for a distinct request type, and
getter methods are provided for each field.
A method to turn the object into a JSON string is also included in the class.
*/
//import the required packages
package blockchaintask1;
import com.google.gson.Gson;
//A public RequestMessage class with the required methods
public class RequestMessage {
   //declare the selection
   String selection;
   //declare enteredDifficulty to store the entered difficulty
   int enteredDifficulty;
   //declare the transaction
   String transaction;
   //declare the corruptID for corruption
   int corruptID;
   //declare the new transaction to corrupt the block
```

```
    String newTransaction;

    /**
     * An Overloaded constructor with selection parameter
     */
    public RequestMessage(String selection){
        this.selection=selection;
    }

    /**
     * An Overloaded constructor with selection, enteredDifficulty, and
transaction parameter to add a block
     */
    public RequestMessage(String selection,int enteredDifficulty,String
transaction){
        this.selection=selection;
        this.enteredDifficulty=enteredDifficulty;
        this.transaction=transaction;
    }

    /**
     * An Overloaded constructor with selection, newTransaction, and corruptID
parameters to corrupt a block
     */
    public RequestMessage(String selection,String newTransaction,int corruptID){
        this.selection=selection;
        this.corruptID=corruptID;
        this.newTransaction=newTransaction;
    }

    /**
     Getter method for enteredDifficulty
     */
    public int getEnteredDifficulty() {
        return enteredDifficulty;
    }

    /**
     Getter method for transaction
     */
    public String getTransaction() {
        return transaction;
    }

    /**
     Getter method for corruptID
     */
    public int getCorruptID() {
        return corruptID;
```

```
    }

    /**
     Getter method for selection
     */
    public String getSelection() {
        return selection;
    }

    /**
     Getter method for newTransaction
     */
    public String getNewTransaction() {
        return newTransaction;
    }

    /**
     Getter method for RequestMessage in JSON format
     */
    public String getRequestMessage(){
        Gson  gson=new Gson();
        //convert the object to JSON
        return gson.toJson(this);
    }

}
```

ResponseMessage class:

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2@andrew.cmu.edu
This class represents a reply message that the blockchain server sends to the
client.
It includes details about the blockchain, including the chain hash, total
number of hashes, total difficulty, most recent nonce, and hash rate per
second.
The class has two constructors: one for the blockchain data and one for the
response message.
The class offers getter methods for each field as well as a method to turn an
object into a JSON string.
*/
//import the required packages
package blockchaintask1;
import com.google.gson.Gson;
//A public ResponseMessage class with the required methods
public class ResponseMessage {
```

```
    //declare a selection string
    String selection;
    //declare size
    int size;
    //declare chainHash
    String chainHash;
    //declare totalHashes
    int totalHashes;
    //declare totalDiff of the chain
    int totalDiff;
    //declare recent nonce of the last block
    int recentNonce;
    //declare diff of the recent block
    int diff;
    //declare hashesPerSecond
    int hps;
    //declare response string
    String response;

    /**
     An Overloaded constructor with the details of the blockchain
     */
    public ResponseMessage(String selection,int size,String chainHash,int
totalHashes,int totalDiff,int recentNonce,int diff,int hps){
        this.selection=selection;
        this.size=size;
        this.chainHash=chainHash;
        this.totalHashes=totalHashes;
        this.totalDiff=totalDiff;
        this.recentNonce=recentNonce;
        this.diff=diff;
        this.hps=hps;
    }

    /**
     An Overloaded constructor with the selection string and response
     */
    public ResponseMessage(String selection,String response){
        this.selection=selection;
        this.response=response;
    }

    /**
     Getter method for selection
     */
    public String getSelection() {
        return selection;
    }
    /**
```

```
     Getter method for size
     */
    public int getSize() {
        return size;
    }

    /**
     Getter method for chainHash
     */
    public String getChainHash() {
        return chainHash;
    }

    /**
     Getter method for totalHashes
     */
    public int getTotalHashes() {
        return totalHashes;
    }

    /**
     Getter method for totalDiff
     */
    public int getTotalDiff() {
        return totalDiff;
    }

    /**
     Getter method for recentNonce
     */
    public int getRecentNonce() {
        return recentNonce;
    }

    /**
     Getter method for diff
     */
    public int getDiff() {
        return diff;
    }

    /**
     Getter method for hps
     */
    public int getHps() {
        return hps;
    }

    /**
```

```
     Getter method for response
     */
    public String getResponse() {
        return response;
    }

    /**
     Getter method for ResponseMessage in JSON format
     */
    public String getResponseMessage()
    {
        Gson gson=new Gson();
        //convert the object to JSON
        return gson.toJson(this);
    }


}
```

Task 1 Server Source Code

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2@andrew.cmu.edu
This is a server program for a blockchain. It listens on a port for incoming
connections and processes requests from the client. The requests are in the
form of JSON strings that are
parsed and processed to execute different operations of blockchain. The
program contains methods to
process different types of requests such as viewing blockchain status, adding
a new block, verifying the entire
chain, viewing the entire chain, corrupting the chain, and repairing the
chain. The program uses the Gson library
to parse the JSON strings and convert Java objects to JSON strings. The
program uses a BlockChain object to
represent the blockchain and its functionalities.
*/
//import the required packages
package blockchaintask1;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
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.util.Scanner;
import static blockchaintask1.BlockChain.getTime;
//Referred my project2Task4 code for TCP
public class BlockChainServer {
   //declare a clientSocket
   static Socket clientSocket = null;
   //declare a ServerSocket
```

```
    static ServerSocket listenSocket=null;
    //declare a port
    static int port;
    //declare a scanner object
    static Scanner scanner;
    //declare a blockChain object to create a chain
    static BlockChain blockChain;
    //declare and initialize index to 0
    static int index=0;
    //declare a genesis block
    static Block genesis;
    //declare a GSON object
    static Gson gson;
    //declare a GSON's JSONObject
    static JsonObject json;
    //declare a ResponseMessage object
    static ResponseMessage responseMessage;
    //declare a RequestMessage to get the message from the  client
    static RequestMessage messageFromClient;

    /**
     * The main method sets up a server socket to start listening on a
user-specified port.
     * The blockChain object is then initialized, the hash rate is calculated,
and the genesis block is created.
     * After that, a while loop starts up and continuously checks for new
client connections.
     * The processRequest method is called by the server after reading the
client's request message during a connection.
     * The response message that has been generated is sent back to the client.
Up to a manual shutdown of the program,
     * this process keeps running.
     */
    public static void main(String[] args) {
        try {
            //create a scanner object
            scanner=new Scanner(System.in);
            //get the port number from the user to listen
            System.out.print("Enter the port number for the server to listen:");
            port = Integer.parseInt(scanner.nextLine());
            //create a ServerSocket object with the port
            listenSocket= new ServerSocket(port);
            System.out.println("Blockchain server running");
            //create new BlockChain object
            blockChain=new BlockChain();
            //compute the hashesPerSecond by the machine
            blockChain.computeHashesPerSecond();
            //create a new genesis block
            genesis=new Block(0,getTime(),"Genesis",2);
```

```
            //add 1 to the index
            index=index+1;
            //set the previousHash to empty string
            genesis.setPreviousHash("");
            //add the genesis block to the chain
            blockChain.addBlock(genesis);
            //infinite loop
            while (true) {
                //accept connections from client
                clientSocket = listenSocket.accept();
                System.out.println("We have a visitor");
                Scanner in;
                //create a new scanner object to get the message from the
client
                in = new Scanner(clientSocket.getInputStream());
                //declare a printWriter onject to send the message to the
client
                PrintWriter out;
                out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
                while (in.hasNextLine())
                {   //get the JSON request message from the client
                    String requestJSON = in.nextLine();
                    //call processRequest() to process the message and do the
operations on blockchain
                    String responseJSON = processRequest(requestJSON);
                    //send the JSON response to the client
                    out.println(responseJSON);
                    out.flush();
                }
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     *processRequest method processes the JSON request message from the client
by first parsing it
     * into a RequestMessage object using the Gson library. The selection value
in the request is
     * used to determine the operation to be performed on the blockchain.
     * The appropriate operation method is then called with the selection value
as an argument.The JSON
     * returned by the called methods is sent back to the main method.
     * @param requestJSON
     * @return JSON response
     */
    public static String processRequest(String requestJSON)
```

```
    {    //create a GSON object
         gson=new Gson();
         //parse the JSON to RequestMessage object named as messageFromClient
         messageFromClient = gson.fromJson(requestJSON,RequestMessage.class);
         //get the selection from the messageFromClient object
         int selection = Integer.parseInt(messageFromClient.getSelection());
         //if the selection is 0 call processViewBlockChainStatus method
         if(selection==0){
             return processViewBlockChainStatus(selection);
         }
         //if the selection is 1 then call processAddBlock method
         if(selection==1){
             return processAddBlock(selection);
         }
         //if the selection is 2 then call processVerifyBlockChain method
         if(selection==2){
             return processVerifyBlockChain(selection);
         }
         //if the selection is 3 then call processViewBlockChain method
         if(selection==3){
             return processViewBlockChain(selection);
         }
         //if the selection is 4 then call processCorruptBlockChain method
         if(selection==4){
             return processCorruptBlockChain(selection);
         }
         //if the selection is 5 then call processRepairBlockChain method
         if(selection==5){
             return processRepairBlockChain(selection);
         }
         //else return null
         return null;
    }

    /**
     * processViewBlockChainStatus method processes the request to view the
current status of the
     * blockchain network, including the number of blocks in the chain, the
current difficulty level,
     * and the latest block attributes etc.
     * @param selection
     * @return JSON response string
     */
    public static String  processViewBlockChainStatus(int selection)
    {
         //create a ResponseMessage object with all the attributes of the
blockchain
         responseMessage=new
ResponseMessage(String.valueOf(selection),blockChain.getChainSize(),blockChain.
```

```
getChainHash(),(int)blockChain.getTotalExpectedHashes(),blockChain.getTotalDiff
iculty(),blockChain.nonceMostRecent().intValue(),
blockChain.difficultyMostRecent(),blockChain.getHashesPerSecond());
        //parse the JSON string returned by the getResponseMessage to
JSONObject
        json = gson.fromJson(responseMessage.getResponseMessage(),
JsonObject.class);
        //create a new JsonObject to put the required attributes
        JsonObject reponseToCLient = new JsonObject();
        //add selection
        reponseToCLient.addProperty("selection",
json.get("selection").getAsString());
        //add size
        reponseToCLient.addProperty("size", json.get("size").getAsInt());
        //add chainHash
        reponseToCLient.addProperty("chainHash",
json.get("chainHash").getAsString());
        //add totalHashes
        reponseToCLient.addProperty("totalHashes",
json.get("totalHashes").getAsInt());
        //add totalDiff
        reponseToCLient.addProperty("totalDiff",
json.get("totalDiff").getAsInt());
        //add recentNonce
        reponseToCLient.addProperty("recentNonce",
json.get("recentNonce").getAsInt());
        //add diff
        reponseToCLient.addProperty("diff", json.get("diff").getAsInt());
        //add hps
        reponseToCLient.addProperty("hps", json.get("hps").getAsInt());
        //print the response format
        System.out.println("Response :" +reponseToCLient);
        //return the JSON string of reponseToCLient
        return reponseToCLient.toString();
    }

    /**
     * processAddBlock parse the JSON request to get the required data to
create a block and add the
     * block to the blockchain through addblock() method . It creates a
ResponseMessage object and parse it to JSON message to send
     * it back to the client
     * @param selection
     * @return JSON response
     */
    public static String processAddBlock(int selection)
    {
        //Adding a block
        System.out.println("Adding a block");
```

```
        //declare difficulty
        int difficulty;
        //declare transaction
        String transaction;
        //get the difficulty from the messageFromClient object
        difficulty=messageFromClient.getEnteredDifficulty();
        //get the transaction from the messageFromClient object
        transaction=messageFromClient.getTransaction();
        //get the start time
        long start = System.currentTimeMillis();
        //create a new block with the data from the client
        Block block= new Block(index,getTime(),transaction,difficulty);
        //increment the index
        index=index+1;
        //add the block to the chain
        blockChain.addBlock(block);
        //get the stop time
        long stop = System.currentTimeMillis();
        //create a response string
        String addResult="Total execution time to add this block was
"+(stop-start)+" milliseconds";
       System.out.println("Setting response to "+addResult);
        //create a ResponseMessage with the selection and addResult response
string
        responseMessage=new
ResponseMessage(String.valueOf(selection),addResult);
        //convert the ResponseMessage object to JSONObject
        json = gson.fromJson(responseMessage.getResponseMessage(),
JsonObject.class);
        //create a new JSONObject to put the required values to send it back to
the client
        JsonObject reponseToCLient = new JsonObject();
        //add selection
        reponseToCLient.addProperty("selection",
json.get("selection").getAsString());
        //add response
        reponseToCLient.addProperty("response",
json.get("response").getAsString());
        //print the response
        System.out.println(reponseToCLient);
        //return the response JSON
        return reponseToCLient.toString();
    }

    /**
     *processVerifyBlockChain method calls blockChain.isChainValid() method to
verify the chain
     * and create a ResponseMessage object with the response string and parse
it to JSON to send it
```

```
     * back to the client
     * @param selection
     * @return JSON string
     */
    public static String processVerifyBlockChain(int selection)
    {
        System.out.println("Verifying entire chain");
        //get the start time
        long start = System.currentTimeMillis();
        //call the isChainValid() method to verify the chain
        String varificationResult = "Chain Verification : "+
blockChain.isChainValid();
        //get the stop time
        long stop = System.currentTimeMillis();
        //create a varificationResult response string
        varificationResult=varificationResult+"\nTotal execution time to verify
the chain was "+(stop-start)+" milliseconds";
        System.out.println(varificationResult);
        System.out.println("Setting response to "+"Total execution time to
verify the chain was "+(stop-start)+" milliseconds");
        //create a ResponseMessage object with the varificationResult string
        responseMessage=new
ResponseMessage(String.valueOf(selection),varificationResult);
        //parse the responseMessage object to a JsonObject
        json = gson.fromJson(responseMessage.getResponseMessage(),
JsonObject.class);
        //create a new JsonObject to put the required values to the JSON
        JsonObject reponseToCLient = new JsonObject();
        //add selection
        reponseToCLient.addProperty("selection",
json.get("selection").getAsString());
        //add response string
        reponseToCLient.addProperty("response",
json.get("response").getAsString());
        //retur the JSON in the string format
        return reponseToCLient.toString();
    }

    /** processViewBlockChain method call the blockChain.toString() to get the
JSON format of the chain.
     * It then create a ResponseMessage object porse it to a JSONObject and
returns the JSON response
     * @param selection
     * @return JSON string
     */
    public static String processViewBlockChain(int selection)
    {
        System.out.println("View the Blockchain");
        //calls to toString() method
```

```
        String viewResult= blockChain.toString();
        System.out.println("Setting response to "+viewResult);
        //create a ResponseMessage object
        responseMessage=new
ResponseMessage(String.valueOf(selection),viewResult);
        //parse it to JSONObject
        json = gson.fromJson(responseMessage.getResponseMessage(),
JsonObject.class);
        //create a new JsonObject object to put the required fields
        JsonObject reponseToCLient = new JsonObject();
        //add selection
        reponseToCLient.addProperty("selection",
json.get("selection").getAsString());
        //add response
        reponseToCLient.addProperty("response",
json.get("response").getAsString());
        //return JSON string
        return reponseToCLient.toString();
    }

    /**
     * processCorruptBlockChain method parses the JSON request to get the
required data to corrput a block.
     * Calls the setData() mehtod to set the new transaction and returns a JSON
response
     * @param selection
     * @return JSON string
     */

    public static String processCorruptBlockChain(int selection)
    {
        System.out.println("corrupt the Blockchain");
        //get the block ID to corrupt
        int ID= messageFromClient.getCorruptID();
        // get the new transaction
        String newData=messageFromClient.getNewTransaction();
        //set the new transaction
        blockChain.getBlock(ID).setData(newData);
        //create a response string called corruptResult
        String corruptResult ="Block "+ID+" now holds "+newData;
        System.out.println(corruptResult);
        //create a new ResponseMessage object with the selection and
corruptResult string
        responseMessage=new
ResponseMessage(String.valueOf(selection),corruptResult);
        //parse it to JSONObject
        json = gson.fromJson(responseMessage.getResponseMessage(),
JsonObject.class);
        //create a new JSONObject to add the required fields
```

```
        JsonObject reponseToCLient = new JsonObject();
        System.out.println("Setting response to "+corruptResult);
        //add selection
        reponseToCLient.addProperty("selection",
json.get("selection").getAsString());
        //add response
        reponseToCLient.addProperty("response",
json.get("response").getAsString());
        //return the JSON string format
        return reponseToCLient.toString();
    }

    /**
     * processRepairBlockChain method call the blockchain's repairChain()
method to perform the
     * repair operation. It then creates a ResponseMessage and parse it to JSON
and return the JSON
     * @param selection
     * @return JSON string
     */
    public static String processRepairBlockChain(int selection)
    {
        System.out.println("Repairing the entire chain");
        //get the start time
        long start = System.currentTimeMillis();
        //call the repairChain() method
        blockChain.repairChain();
        //get the stop time
        long stop = System.currentTimeMillis();
        //create a repairMessage string to set the reponse
        String repairMessage= "Total execution time to repair the chain was
"+(stop-start)+" milliseconds";
        //create a ResponseMessage onject with the selection and repairMessage
string
        responseMessage=new
ResponseMessage(String.valueOf(selection),repairMessage);
        //parse it tp JSONObject
        json = gson.fromJson(responseMessage.getResponseMessage(),
JsonObject.class);
        //create a new JSONObject to add the required fields
        JsonObject reponseToCLient = new JsonObject();
        System.out.println("Setting response to "+repairMessage);
        //add selection
        reponseToCLient.addProperty("selection",
json.get("selection").getAsString());
        //add response
        reponseToCLient.addProperty("response",
json.get("response").getAsString());
        //return the JSON string
```

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

RequestMessage class:

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2
The request message that the client sends to the blockchain server is
represented by this class.
It includes details on the kind of request being performed.
The class comprises three constructors, each for a distinct request type, and
getter methods are provided for each field.
A method to turn the object into a JSON string is also included in the class.
*/
//import the required packages
package blockchaintask1;
import com.google.gson.Gson;
//A public RequestMessage class with the required methods
public class RequestMessage {
    //declare the selection
    String selection;
    //declare enteredDifficulty to store the entered difficulty
    int enteredDifficulty;
    //declare the transaction
    String transaction;
    //declare the corruptID for corruption
    int corruptID;
    //declre the new transaction to corrupt the block
    String newTransaction;

    /**
     * An Overloaded constructor with selection parameter
     */
    public RequestMessage(String selection){
        this.selection=selection;
    }

    /**
     * An Overloaded constructor with selection, enteredDifficulty, and
transaction parameter to add a block
     */
    public RequestMessage(String selection,int enteredDifficulty,String
transaction){
        this.selection=selection;
        this.enteredDifficulty=enteredDifficulty;
```

```
        this.transaction=transaction;
    }

    /**
     * An Overloaded constructor with selection, newTransaction, and corruptID
parameters to corrupt a block
     */
    public RequestMessage(String selection,String newTransaction,int corruptID){
        this.selection=selection;
        this.corruptID=corruptID;
        this.newTransaction=newTransaction;
    }

    /**
     Getter method for enteredDifficulty
     */
    public int getEnteredDifficulty() {
        return enteredDifficulty;
    }

    /**
     Getter method for transaction
     */
    public String getTransaction() {
        return transaction;
    }

    /**
     Getter method for corruptID
     */
    public int getCorruptID() {
        return corruptID;
    }

    /**
     Getter method for selection
     */
    public String getSelection() {
        return selection;
    }

    /**
     Getter method for newTransaction
     */
    public String getNewTransaction() {
        return newTransaction;
    }

    /**
```

```
     Getter method for RequestMessage in JSON format
     */
    public String getRequestMessage(){
        Gson  gson=new Gson();
        //convert the object to JSON
        return gson.toJson(this);
    }

}
```

ResponseMessage class:

```
/**
* Author: Praveen Ramesh
* Andrew ID: pramesh2
This class represents a reply message that the blockchain server sends to the
client.
It includes details about the blockchain, including the chain hash, total
number of hashes, total difficulty, most recent nonce, and hash rate per
second.
The class has two constructors: one for the blockchain data and one for the
response message.
The class offers getter methods for each field as well as a method to turn an
object into a JSON string.
*/
//import the required packages
package blockchaintask1;
import com.google.gson.Gson;
//A public ResponseMessage class with the required methods
public class ResponseMessage {
    //declare a selection string
    String selection;
    //declare size
    int size;
    //declare chainHash
    String chainHash;
    //declare totalHashes
    int totalHashes;
    //declare totalDiff of the chain
    int totalDiff;
    //declare recent nonce of the last block
    int recentNonce;
    //declare diff of the recent block
    int diff;
    //declare hashesPerSecond
```

```
    int hps;
    //declare response string
    String response;

    /**
     An Overloaded constructor with the details of the blockchain
     */
    public ResponseMessage(String selection,int size,String chainHash,int
totalHashes,int totalDiff,int recentNonce,int diff,int hps){
        this.selection=selection;
        this.size=size;
        this.chainHash=chainHash;
        this.totalHashes=totalHashes;
        this.totalDiff=totalDiff;
        this.recentNonce=recentNonce;
        this.diff=diff;
        this.hps=hps;
    }

    /**
     An Overloaded constructor with the selection string and response
     */
    public ResponseMessage(String selection,String response){
        this.selection=selection;
        this.response=response;
    }

    /**
     Getter method for selection
     */
    public String getSelection() {
        return selection;
    }
    /**
     Getter method for size
     */
    public int getSize() {
        return size;
    }

    /**
     Getter method for chainHash
     */
    public String getChainHash() {
        return chainHash;
    }

    /**
     Getter method for totalHashes
```

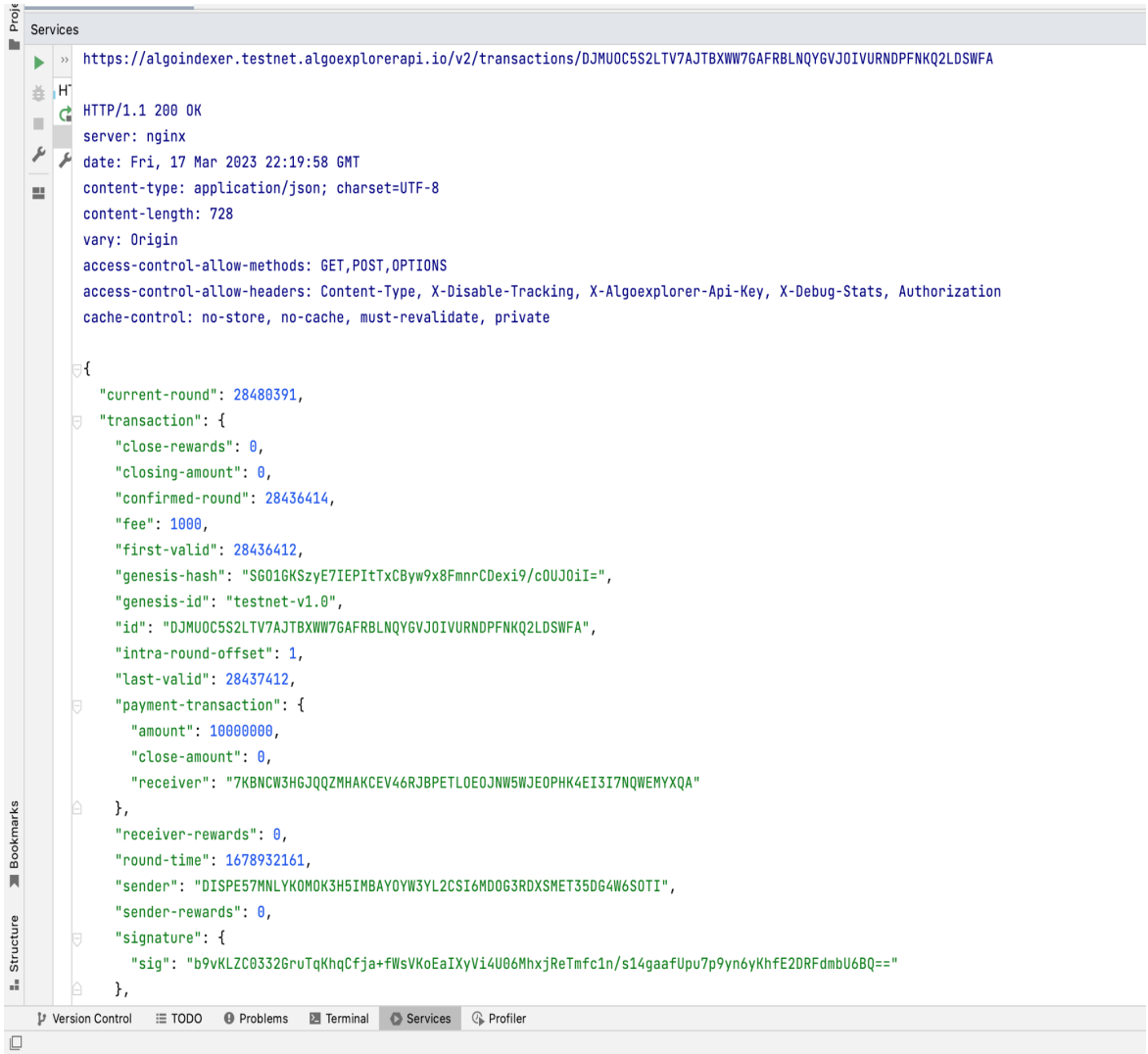
```
 */
public int getTotalHashes() {
    return totalHashes;
}

/**
 Getter method for totalDiff
 */
public int getTotalDiff() {
    return totalDiff;
}

/**
 Getter method for recentNonce
 */
public int getRecentNonce() {
    return recentNonce;
}

/**
 Getter method for diff
 */
public int getDiff() {
    return diff;
}

/**
 Getter method for hps
 */
public int getHps() {
    return hps;
}

/**
 Getter method for response
 */
public String getResponse() {
    return response;
}

/**
 Getter method for ResponseMessage in JSON format
 */
public String getResponseMessage()
{
    Gson gson=new Gson();
    //convert the object to JSON
    return gson.toJson(this);
}
```

}

Project3Task2

Adding fund transaction GET request:

Genesis transaction response:



Text version of response:
https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/DJMUOC5S2LTV7AJTBXWW7GAFRBLNQYGVJOIVURNDPFNKQ2LDSWFA

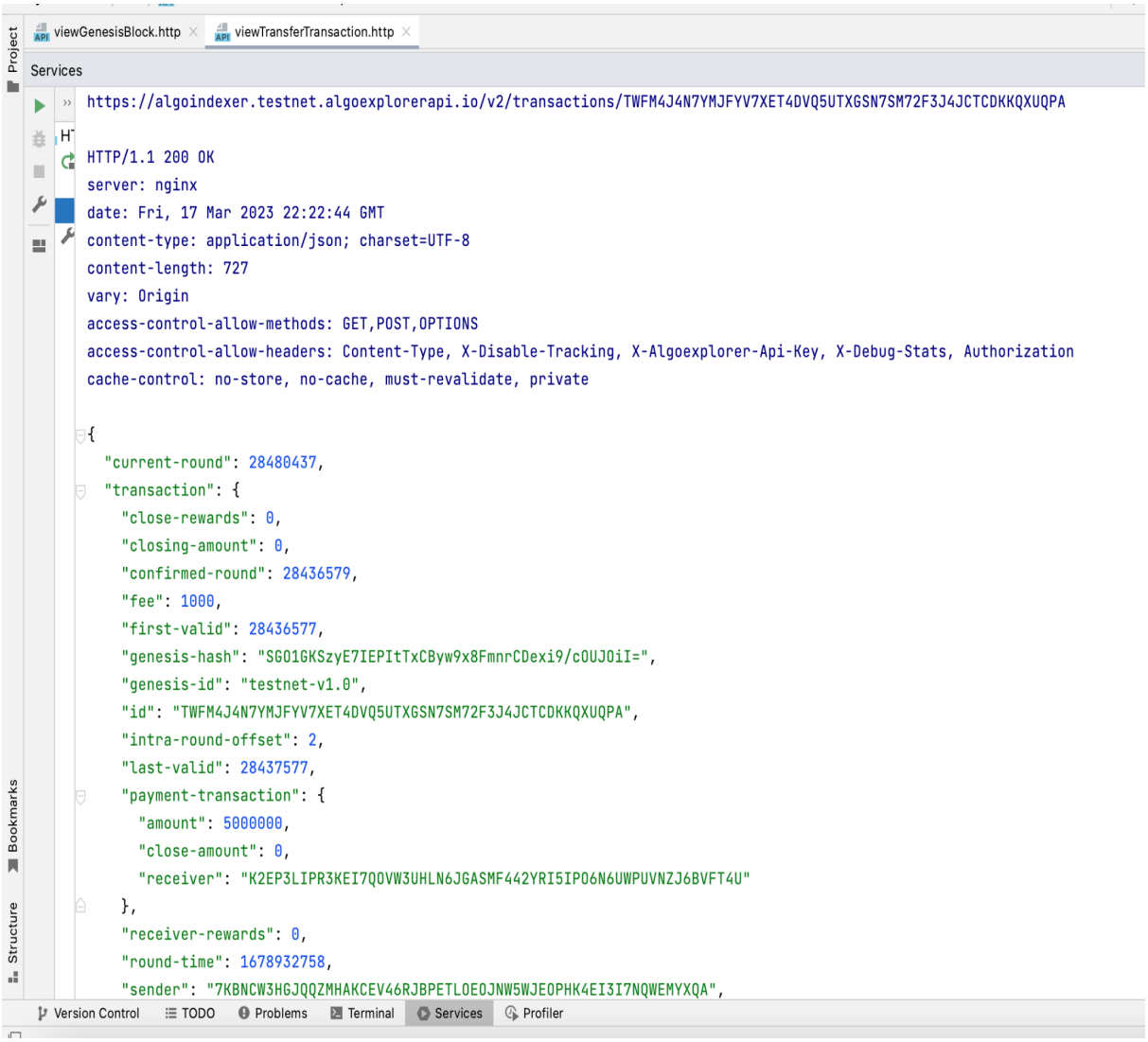
HTTP/1.1 200 OK
server: nginx
date: Fri, 17 Mar 2023 07:03:09 GMT
content-type: application/json; charset=UTF-8
content-length: 728
vary: Origin
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-Stats, Authorization
cache-control: no-store, no-cache, must-revalidate, private


```
{
  "current-round": 28465220,
  "transaction": {
    "close-rewards": 0,
    "closing-amount": 0,
    "confirmed-round": 28436414,
    "fee": 1000,
    "first-valid": 28436412,
    "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "genesis-id": "testnet-v1.0",
    "id": "DJMUOC5S2LTV7AJTBXWW7GAFRBLNQYGVJOIVURNDPFNKQ2LDSWFA",
    "intra-round-offset": 1,
    "last-valid": 28437412,
    "payment-transaction": {
      "amount": 10000000,
      "close-amount": 0,
      "receiver": "7KBNCW3HGJQQZMHAKCEV46RJBPETLOEOJNW5WJEOPHK4EI3I7NQWEMYXQA"
    },
    "receiver-rewards": 0,
    "round-time": 1678932161,
    "sender": "DISPE57MNLYKOMOK3H5IMBAYOYW3YL2CSI6MDOG3RDXSMET35DG4W6SOTI",
    "sender-rewards": 0,
    "signature": {
      "sig":
"b9vKLZC0332GruTqKhqCfja+fWsVKoEaIXyVi4U06MhxjReTmfc1n/s14gaafUpu7p9yn6yKhfE2DRFdmbU6BQ=="
    },
    "tx-type": "pay"
  }
}
```

Response file saved.
> 2023-03-17T030309.200.json
Response code: 200 (OK); Time: 523ms (523 ms); Content length: 728 bytes (728 B)

Fund transfer(5 Algos) transaction GET request:

Fund transfer(5 Algos) response:



Text version of response:
https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/TWFM4J4N7YMJFYV7XET4DVQ5UTXGSN7SM72F3J4JCTCDKKQXUQPA

HTTP/1.1 200 OK
server: nginx
date: Fri, 17 Mar 2023 07:07:28 GMT
content-type: application/json; charset=UTF-8
content-length: 727
vary: Origin
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-Stats, Authorization
cache-control: no-store, no-cache, must-revalidate, private


```
{
  "current-round": 28465291,
  "transaction": {
    "close-rewards": 0,
    "closing-amount": 0,
    "confirmed-round": 28436579,
    "fee": 1000,
    "first-valid": 28436577,
    "genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "genesis-id": "testnet-v1.0",
    "id": "TWFM4J4N7YMJFYV7XET4DVQ5UTXGSN7SM72F3J4JCTCDKKQXUQPA",
    "intra-round-offset": 2,
    "last-valid": 28437577,
    "payment-transaction": {
      "amount": 5000000,
      "close-amount": 0,
      "receiver": "K2EP3LIPR3KEI7QOVW3UHLN6JGASMF442YRI5IPO6N6UWPUVNZJ6BVFT4U"
    },
    "receiver-rewards": 0,
    "round-time": 1678932758,
    "sender": "7KBNCW3HGJQQZMHAKCEV46RJBPETLOEOJNW5WJEOPHK4EI3I7NQWEMYXQA",
    "sender-rewards": 0,
    "signature": {
      "sig":
"f1W3qVd50c9WxG9PaIS/aE9pKu3R1Yh2MCzP/C6H/7MXle8zNQdzl9pMdngeEg1YLIUM7WjZRcZjgVk7z6kgAQ=="
    },
    "tx-type": "pay"
  }
}
```

Response file saved.
> 2023-03-17T030728.200.json
Response code: 200 (OK); Time: 416ms (416 ms); Content length: 727 bytes (727 B)