Project 3
Aditi Gupta – argupta@andrew.cmu.edu

Project3Task0

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.
Enter a number between 0 and 6:
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: 2544529
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 118
Chain hash:
00DA8CEAA234CF34A71006E8583E8643C8B4F8DB14A202DECF4F4BFAB4BA4659
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.
Enter a number between 0 and 6:
1
Enter difficulty > 0
2
Enter transaction
Mike pays Marty 100 DSCoin
Total execution time to add this block was 10.0 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.
Enter a number between 0 and 6:
1
Enter difficulty > 0
2
Enter transaction
Marty pays Joe 50 DSCoin
Total execution time to add this block was 3.0 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.
Enter a number between 0 and 6:
1
Enter difficulty > 0
2
Enter transaction
Joe pays Andy 10 DS Coin
Total execution time to add this block was 1.0 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.
Enter a number between 0 and 6:
2
Chain verification: TRUE
Total execution time required to verify the chain was 1.0 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.
Enter a number between 0 and 6:
3
View the Blockchain
{"ds_chain" : [{"index" : 0,"time stamp " : "2023-10-25 14:21:32.848","Tx " :
"Genesis","PrevHash" : "","nonce" : 118,"difficulty": 2},
{"index" : 1,"time stamp " : "2023-10-25 14:29:42.991","Tx " : "Mike pays Marty 100
DSCoin","PrevHash" :
"00DA8CEAA234CF34A71006E8583E8643C8B4F8DB14A202DECF4F4BFAB4BA4659","no
nce" : 372,"difficulty": 2},
{"index" : 2,"time stamp " : "2023-10-25 14:29:51.792","Tx " : "Marty pays Joe 50
DSCoin","PrevHash" :
"0033915928641076C96FB7DD5B50973B738CD1236E6575B5A8FCD14102AC31A9","nonce
" : 117,"difficulty": 2},
{"index" : 3,"time stamp " : "2023-10-25 14:30:03.237","Tx " : "Joe pays Andy 10 DS
Coin","PrevHash" :
"0016397426CE2ECFC93373863FA35057D59F273C269A6F25C95D4F084C4F8C2D","nonce
" : 7,"difficulty": 2}
 ], "chainHash" :
"00035BF443F8B4FD833D324DC21C01BCC07C9D12348C0BE3FD070D625200A22A"}
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.
Enter a number between 0 and 6:
4
Corrupt the Blockchain
Enter block ID of block to Corrupt:
1
Enter new data for block 1
Mike pays Marty 76 DSCoin
Block 1 now holds Mike pays Marty 76 DSCoin
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
Enter a number between 0 and 6:
3
View the Blockchain
{"ds_chain" : [{"index" : 0,"time stamp " : "2023-10-25 14:21:32.848","Tx " :
"Genesis","PrevHash" : "","nonce" : 118,"difficulty": 2},
{"index" : 1,"time stamp " : "2023-10-25 14:29:42.991","Tx " : "Mike pays Marty 76
DSCoin","PrevHash" :
"00DA8CEAA234CF34A71006E8583E8643C8B4F8DB14A202DECF4F4BFAB4BA4659","no
nce" : 372,"difficulty": 2},
{"index" : 2,"time stamp " : "2023-10-25 14:29:51.792","Tx " : "Marty pays Joe 50
DSCoin","PrevHash" :
"0033915928641076C96FB7DD5B50973B738CD1236E6575B5A8FCD14102AC31A9","nonce
" : 117,"difficulty": 2},
{"index" : 3,"time stamp " : "2023-10-25 14:30:03.237","Tx " : "Joe pays Andy 10 DS
Coin","PrevHash" :
"0016397426CE2ECFC93373863FA35057D59F273C269A6F25C95D4F084C4F8C2D","nonce
" : 7,"difficulty": 2}
 ], "chainHash" :
"00035BF443F8B4FD833D324DC21C01BCC07C9D12348C0BE3FD070D625200A22A"}
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.
Enter a number between 0 and 6:
2
Improper hash on node 1 Does not begin with 00
Chain verification: FALSE
Total execution time required to verify the chain was 1.0 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.
Enter a number between 0 and 6:
5
Repairing the entire chain
Total execution time required to repair the chain was 9.0 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.
Enter a number between 0 and 6:
2
Chain verification: TRUE
Total execution time required to verify the chain was 2.0 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.
Enter a number between 0 and 6:
1
Enter difficulty > 0
4
Enter transaction
Andy pays Sean 25 DSCoin
Total execution time to add this block was 39.0 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.
Enter a number between 0 and 6:
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: 2544529
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 5482
Chain hash:
0000F3C78652993C7E614F7BA6D1BFA27881951C4F78912D126A5B8D5BA76943
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.
Enter a number between 0 and 6:
1
Enter difficulty > 0
5
Enter transaction
Aditi pays Marty 100 DSCoin
Total execution time to add this block was 1432.0 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.
Enter a number between 0 and 6:
6

Process finished with exit code 0
```

Task 0 Block.java

```java
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/Block.html
// Taken code from https://www.baeldung.com/java-blockchain
// Project 3 Task 0

package blockchaintask0;

import com.google.gson.Gson;
import com.google.gson.annotations.SerializedName;

import java.math.BigInteger;

/**
 * This class represents a simple Block.
 *
 * Each Block object has an index - the position of the block on the chain.
 * The first block (the so called Genesis block) has an index of 0.
 *
 * Each block has a timestamp - a Java Timestamp object, it holds the time of
the block's creation.
 *
 * Each block has a field named data - a String holding the block's single
transaction details.
 *
 * Each block has a String field named previousHash - the SHA256 hash of a
block's parent.
 * This is also called a hash pointer.
 *
 * Each block holds a nonce - a BigInteger value determined by a proof of
work routine.
 * This has to be found by the proof of work logic.
 * It has to be found so that this block has a hash of the proper difficulty.
 * The difficulty is specified by a small integer representing the minimum
number of leading hex zeroes the hash must have.
 *
 * Each block has a field named difficulty - it is an int that specifies the
minimum number of left most hex digits needed by a proper hash.
 * The hash is represented in hexadecimal. If, for example, the difficulty is
3, the hash must have at least three leading hex 0's (or,1 and 1/2 bytes).
 * Each hex digit represents 4 bits.
 */
```

```java
public class Block extends java.lang.Object{
    private int index;
    private java.sql.Timestamp timestamp;
    @SerializedName(value = "Tx")
    private java.lang.String data;
    @SerializedName(value = "PrevHash")
    private java.lang.String previousHash;
    private BigInteger nonce = BigInteger.ZERO;
    private int difficulty;

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

    /**
     * This method computes a hash of the concatenation of the index,
timestamp, data,
     * previousHash, nonce, and difficulty.
     *
     * Returns: a String holding Hexadecimal characters
     */

    public java.lang.String calculateHash() throws Exception {
        String total = "";
        total+=String.valueOf(this.index);

        // Code taken from https://www.tutorialspoint.com/java-sql-timestamp-
tostring-method-with-
example#:~:text=The%20toString()%20method%20of,Timestamp%20object%20to%20a%20
String.
        total+=this.timestamp.toString();
        total+=this.data;
        total+=this.previousHash;
        total+=String.valueOf(this.nonce);
        total+=String.valueOf(this.difficulty);
        String hash=Hash.hash(total);
        return hash;
    }

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

/**
 * The proof of work methods finds a good hash. It increments the nonce until
it produces a good hash.
 *
 * This method calls calculateHash() to compute a hash of the concatenation
of the index, timestamp, data,
 * previousHash, nonce, and difficulty. If the hash has the appropriate
number of leading hex zeroes,
 * it is done and returns that proper hash.
 * If the hash does not have the appropriate number of leading hex zeroes,
 * it increments the nonce by 1 and tries again. It continues this process,
burning electricity and CPU cycles,
 * until it gets lucky and finds a good hash.
 *
 * Returns:
 * a String with a hash that has the appropriate number of leading hex
zeroes.
 * The difficulty value is already in the block. This is the minimum number
of hex 0's a proper hash must have.
 */

    // Taken  code from https://stackoverflow.com/questions/66649182/not-
able-to-get-4-leading-zeros-in-sha256-hash-proof-of-work-ever-java
    public java.lang.String proofOfWork() throws Exception {
        String hash = "";

        //This line of code suggested by GitHub Copilot
        while(!hash.startsWith("0".repeat(this.difficulty))){
            this.nonce = this.nonce.add(BigInteger.ONE);
            hash = calculateHash();
        }
        return hash;
    }

    /**
     * Simple getter method
     * Returns: difficulty of this block
     */

    public int getDifficulty() {
        return this.difficulty;
    }

    /**
     * Simple setter method
     *
     * Parameters:
     * difficulty - determines how much work is required to produce a proper
hash
     */
    public void setDifficulty(int difficulty) {
```

```java
        this.difficulty = difficulty;
    }

    /**Override Java's toString method
     * Overrides:
     * toString in class java.lang.Object
     * Returns:
     * A JSON representation of all of this block's data is returned.
        */
    public java.lang.String toString(){
        // Create a Gson object
        Gson gson = new Gson();
        // Serialize to JSON
        return gson.toJson(this);
    }


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

    /**Simple getter method

     Returns:
     previous hash
     */
    public java.lang.String getPreviousHash(){
        return this.previousHash;
    }

    /**
     * Simple getter method
     * Returns: index of block
     */
    public int getIndex(){
        return this.index;
    }

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

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

    /**
     * Simple getter method
     * Returns:
     * timestamp of this block
     */
    public java.sql.Timestamp getTimestamp(){
        return this.timestamp;
    }

    /**
     * Simple getter method
     * Returns:
     * this block's transaction
     */
    public java.lang.String getData(){
        return this.data;
    }

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

    public static void main(java.lang.String[] args){
    }

}
```

Task 0 BlockChain.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html
// Taken code from https://www.baeldung.com/java-blockchain
// Project 3 Task 0

package blockchaintask0;


import java.util.ArrayList;

public class BlockChain
        extends java.lang.Object {
     /**
     This BlockChain has exactly three instance members -
      an ArrayList to hold Blocks and a chain hash to hold a SHA256 hash of
the most recently added Block.
      It also maintains an instance variable holding the approximate number
```

```
of hashes per second on this computer.
      This constructor creates an empty ArrayList for Block storage.
      This constructor sets the chain hash to the empty string and sets
hashes per second to 0.
*/
     private ArrayList<Block> blocks;      // To store blocks
    private String chainHash;             // To store the SHA256 hash of the
most recently added block
    private long hashesPerSecond;         // To store the approximate number
of hashes per second on this computer

    public BlockChain() {
        this.blocks = new ArrayList<>();
        this.chainHash = "";
        this.hashesPerSecond = 0;
    }

    /**
     * public java.lang.String getChainHash()
     * Returns:
     * the chain hash.
     */
    public java.lang.String getChainHash(){
        return chainHash;
    }

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

    /**public blockchaintask0.Block getLatestBlock()
     Returns:
     a reference to the most recently added Block.
     */
    public Block getLatestBlock() {
        return blocks.get(blocks.size() - 1);
    }

    /**
     * computeHashesPerSecond
     * public void computeHashesPerSecond()
     * This method computes exactly 2 million hashes and times how long that
process takes.
     * So, hashes per second is approximated as (2 million / number of
seconds).
     * It is run on start up and sets the instance variable hashesPerSecond.
     * It uses a simple string - "00000000" to hash.
     */
    public void computeHashesPerSecond() throws Exception {
        Long startTime = System.currentTimeMillis();
        int numHash = 0;
        while (numHash < 2000000) {
            Hash.hash("00000000");
            numHash++;
        }
        Long endTime = System.currentTimeMillis();
        this.hashesPerSecond =  (int)(numHash / ((endTime - startTime) /
1000.0));
    }
    /**
     * public int getHashesPerSecond()
     * get hashes per second
     * Returns:
     * the instance variable approximating the number of hashes per second.
     */
    public int getHashesPerSecond(){
        return (int)hashesPerSecond;
    }

    /**
     * A new Block is being added to the BlockChain.
     * This new block's previous hash must hold the hash
     * of the most recently added block. After this call on addBlock,
     * the new block becomes the most recently added block on the BlockChain.
     * The SHA256 hash of every block must exhibit proof of work,
     * i.e., have the requisite number of leftmost 0's defined by its
difficulty.
     * Suppose our new block is x. And suppose the old blockchain was
     * a <-- b <-- c <-- d then the chain after addBlock completes is
     * a <-- b <-- c <-- d <-- x. Within the block x, there is a previous
hash field.
     * This previous hash field holds the hash of the block d.
     * The block d is called the parent of x.
     * The block x is the child of the block d.
     * It is important to also maintain a hash of the most recently added
block in a chain hash.
     * Let's look at our two chains again. a <-- b <-- c <-- d.
     * The chain hash will hold the hash of d. After adding x,
     * we have a <-- b <-- c <-- d <-- x. The chain hash now holds the hash
of x.
     * The chain hash is not defined within a block but is defined within the
block chain.
     * The arrows are used to describe these hash pointers.
     * If b contains the hash of a then we write a <-- b.
     * Parameters:
     * newBlock is added to the BlockChain as the most recent block
     */
    public void addBlock(blockchaintask0.Block newBlock) throws Exception {
        newBlock.setPreviousHash(this.chainHash);
        this.blocks.add(newBlock);
        this.chainHash = newBlock.proofOfWork();
    }

    /**
     * public java.lang.String toString()
     * This method uses the toString method defined on each individual block.
     * Overrides:
     * toString in class java.lang.Object
     * Returns:
     * a String representation of the entire chain is returned.
     */

    /**
     * View the Blockchain
     * {"ds_chain" : [ {"index" : 0,"time stamp " : "2022-02-25
17:41:11.927","Tx ": "Genesis","PrevHash" : "","nonce" : 286,"difficulty":
2},
     * {"index" : 1,"time stamp " : "2022-02-25 17:42:46.053","Tx ": "Mike
pays Marty 100 DSCoin","PrevHash" :
"0026883909AA470264145129F134489316E6A38439240D0468D69AA9665D4993","nonce" :
165,"difficulty": 2},
     * {"index" : 2,"time stamp " : "2022-02-25 17:44:27.43","Tx ": "Marty
pays Joe 50 DSCoin","PrevHash" :
"000D14B83028DD36BD6330B8DAB185012F8625E9C9D1A8704E9C1189FD98D9DF","nonce" :
819,"difficulty": 2},
     * {"index" : 3,"time stamp " : "2022-02-25 17:45:22.044","Tx ": "Joe
pays Andy 10 DSCoin","PrevHash" :
"00B4CC539C5CC36AE2F09CC7B857A1330D2D02C00CA90D4A34ACD7E01D7225FC","nonce" :
167,"difficulty": 2}
     */
    //override toString method to give above output
    @Override
    public String toString() {

        StringBuilder sb = new StringBuilder();
        sb.append("{\"ds_chain\" : [");

        for (int i = 0; i < blocks.size(); i++) {
            sb.append(blocks.get(i).toString());  // This will serialize each
block to JSON

            if (i != blocks.size() - 1) {  // if not the last block, append a
comma
                sb.append(",");
            }
            //Code from https://stackoverflow.com/questions/14534767/how-to-
append-a-newline-to-stringbuilder - for reference
            sb.append("\n");
        }
        //Code from https://stackoverflow.com/questions/22682016/how-to-use-
string-builder-to-append-double-quotes-and-or-between-elements
        sb.append(" ], \"chainHash\" :
").append("\"").append(this.chainHash).append("\"").append("}");
        return sb.toString();
    }

    /**
     * public blockchaintask0.Block getBlock(int i)
     * return block at position i
     * Parameters:
     * i -
     * Returns:
     * block at postion i
     */
    public blockchaintask0.Block getBlock(int i){
        return blocks.get(i);
    }
```

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

    /**
    * public double getTotalExpectedHashes()
    * Compute and return the expected number of hashes required for the
entire chain.
    * Returns:
    * totalExpectedHashes
    */
    public double getTotalExpectedHashes(){
        double totalExpectedHashes = 0;
        for (Block block : blocks) {
            //Taken code from
https://bitcoin.stackexchange.com/questions/4565/calculating-average-number-
of-hashes-tried-before-hitting-a-valid-block
            totalExpectedHashes += Math.pow(16, block.getDifficulty());
        }
        return totalExpectedHashes;
    }

    /**
     * public java.lang.String isChainValid()
     * If the chain only contains one block, the genesis block at position 0,
     * this routine computes the hash of the block and
     * checks that the hash has the requisite number of leftmost 0's (proof
of work) as specified in the difficulty field.
     * It also checks that the chain hash is equal to this computed hash.
     * If either check fails, return an error message. Otherwise, return the
string "TRUE".
     * If the chain has more blocks than one, begin checking from block one.
     * Continue checking until you have validated the entire chain.
     * The first check will involve a computation of a hash in Block 0 and a
comparison with the hash pointer in Block 1.
     * If they match and if the proof of work is correct, go and visit the
next block in the chain.
     * At the end, check that the chain hash is also correct.
     * Returns:
     * "TRUE" if the chain is valid, otherwise return a string with an
appropriate error message
     */

    // https://www.baeldung.com/java-blockchain - for reference
    public java.lang.String isChainValid() throws Exception {
        if (blocks.size() == 1) {
            Block block = blocks.get(0);
            String hash = block.calculateHash();
            if (hash.startsWith("0".repeat(block.getDifficulty()))) {
                if (hash.equals(this.chainHash)) {
                    return "TRUE";
                } else {
                    System.out.println("Improper hash on node 0");
                    return "FALSE";
                }
            } else {
                System.out.println("Does not begin with " +
("0".repeat(block.getDifficulty())));
                return "FALSE";
            }
        } else {
            for (int i = 1; i < blocks.size(); i++) {
                Block block = blocks.get(i);
                Block prevBlock = blocks.get(i - 1);
                String hash = block.calculateHash();
                if (!hash.startsWith("0".repeat(block.getDifficulty()))
|| !block.getPreviousHash().equals(prevBlock.calculateHash())){
                        System.out.println("Improper hash on node " + i + "
Does not begin with " + ("0".repeat(block.getDifficulty())));
                        return "FALSE";
                    }
            }
            return "TRUE";
        }
    }

    /**
     * public void repairChain()
     * This routine repairs the chain.
     * It checks the hashes of each block and ensures that any illegal hashes
are recomputed.
     * After this routine is run, the chain will be valid.
     * The routine does not modify any difficulty values.
     * It computes new proof of work based on the difficulty specified in the
Block.
     */
    public void repairChain() throws Exception {
        String hash = "";
        for (int i = 1; i < this.blocks.size(); i++) {
            Block currentBlock = this.blocks.get(i);
            Block prevBlock = this.blocks.get(i - 1);
            //Setting all the hashes again so that the chain is repaired
            currentBlock.setPreviousHash(prevBlock.calculateHash());

            // Getting the proof of work for the current block
            hash=currentBlock.proofOfWork();
        }
        //Assigning the proof of work of the final block to the chain hash
        this.chainHash = hash;
    }

    /**
```

```
     * public static void main(java.lang.String[] args)
     * This routine acts as a test driver for your Blockchain.
     * It will begin by creating a BlockChain object and then adding the
Genesis block to the chain.
     * The Genesis block will be created with an empty string as the pervious
hash and a difficulty of 2.
     * On start up, this routine will also establish the hashes per second
instance member.
     * All blocks added to the Blockchain will have a difficulty passed in to
the program by the user at run time.
     * All hashes will have the proper number of zero hex digits representing
the most significant nibbles in the hash.
     * A nibble is 4 bits. If the difficulty is specified as 3, then all
hashes will begin with 3 or more 0 hex digits (or 3 nibbles, or 12 zero
bits).
     *
     * It is menu driven and will continously provide the user with seven
options:
     * Block Chain Menu
     * 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.
     *
     * If the user selects option 0, the program will display:
     * The number of blocks on the chain
     * Difficulty of most recent block
     * The total difficulty for all blocks Approximate hashes per second on
this machine.
     * Expected total hashes required for the whole chain.
     * The computed nonce for most recent block.
     * The chain hash (hash of the most recent block).
     *
     * If the user selects option 1,
     * the program will prompt for and then read the difficulty level for
this block.
     * It will then prompt for and then read a line of data from the user
(representing a transaction).
     * The program will then add a block containing that transaction to the
block chain.
     * The program will display the time it took to add this block.
     * Note: The first block added after Genesis has index 1.
     * The second has 2 and so on. The Genesis block is at position 0.
     *
     * If the user selects option 2, then call the isChainValid method and
display the results.
     * It is important to note that this method will execute fast.
     * Blockchains are easy to validate but time consuming to modify.
     * Your program needs to display the number of milliseconds it took for
validate to run.
     *
     * If the user selects option 3, display the entire Blockchain contents
as a correctly formed JSON document.
     * See www.json.org.
     *
     * If the user selects option 4, she wants to corrupt the chain.
     * Ask her for the block index (0..size-1) and ask her for the new data
that will be placed in the block.
     * Her new data will be placed in the block. At this point, option 2
(verify chain) should show false.
     * In other words, she will be making a data change to a particular block
and the chain itself will become invalid.
     *
     * If the user selects 5,
     * she wants to repair the chain.
     * That is, she wants to recompute the proof of work for each node that
has become invalid - due perhaps,
     * to an earlier selection of option 4. The program begins at the Genesis
block and checks each block in turn.
     * If any block is found to be invalid, it executes repair logic.
     *
     * Important:
     *
     * Within your comments in the main routine, you must describe how this
system behaves as the difficulty increases.
     * Run some experiments by adding new blocks with increasing
difficulties.
     * Describe what you find. Be specific and quote some times.
     *
     * You need not employ a system clock.
     * You should be able to make clear statements describing the approximate
run times
     * associated with addBlock(), isChainValid(), and chainRepair().
     *
     * Parameters:
     * args - is unused
     */
    public static void main(java.lang.String[] args) throws Exception {
        Long startTime, endTime;
        double elapsedTimeInMilliSeconds;
        BlockChain blockChain = new BlockChain();
        Block genesisBlock = new Block(0, blockChain.getTime(), "Genesis",
2);
        blockChain.addBlock(genesisBlock);
        blockChain.computeHashesPerSecond();
        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 corruption by repairing the
chain.");
            System.out.println("6. Exit.");
            System.out.println("Enter a number between 0 and 6:");
            java.util.Scanner scanner = new java.util.Scanner(System.in);
            int input = scanner.nextInt();
            while (input != 7) {
                if (input == 0) {
                    System.out.println("Current size of chain: " +
blockChain.blocks.size());
```

```java
                    System.out.println("Difficulty of most recent block: " +
blockChain.getLatestBlock().getDifficulty());
                    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.getLatestBlock().getNonce());
                    System.out.println("Chain hash: " +
blockChain.getChainHash());
                    break;
                } else if (input == 1) {
                    System.out.println("Enter difficulty > 0");
                    int difficulty = scanner.nextInt();
                    System.out.println("Enter transaction");

                    //After nextInt only the first word is read, so we need
to use nextLine to read the entire line
                    //Code taken from
https://stackoverflow.com/questions/39514730/how-to-take-input-as-string-
with-spaces-in-java-using-scanner
                    scanner.nextLine();
                    String transaction = scanner.nextLine();
                    Block block = new Block(blockChain.blocks.size(),
blockChain.getTime(), transaction, difficulty);
                     startTime = System.currentTimeMillis();
                    blockChain.addBlock(block);
                     endTime = System.currentTimeMillis();
                     elapsedTimeInMilliSeconds = (endTime - startTime);
                    System.out.println("Total execution time to add this
block was " + elapsedTimeInMilliSeconds + " milliseconds");
                    break;

                    // Time required to add a block increases as the
difficulty increases.
                    // It increases exponentially as the difficulty
increases.
                    // From difficulty 2 to 4 the increase in time required
is less (in magnitude of 10s).
                    // But from difficulty 4 to 5 the increase in time
required is high (in the magnitude of 1000s).
                    // And from difficulty 5 to 6 the increase in time
required is even higher (in the magnitude of 10000s).

                    // For instance:
                    // Time required to add a block of difficulty 2 was
between 1-10 milliseconds
                    // Time required to add a block of difficulty 3 was 27
milliseconds
                    // Time required to add a block of difficulty 4 was 39
milliseconds
                    // Time required to add a block of difficulty 5 was 1432
milliseconds
                    // Time required to add a block of difficulty 6 was 18296
milliseconds
```

```
                } else if (input == 2) {
                     startTime = System.currentTimeMillis();
                    System.out.println("Chain verification: " +
blockChain.isChainValid());
                     endTime = System.currentTimeMillis();
                     elapsedTimeInMilliSeconds = (endTime - startTime);
                    System.out.println("Total execution time required to
verify the chain was " + elapsedTimeInMilliSeconds + " milliseconds");
                    break;
                    // Time required to verify the chain of any length and
any difficulty was between 1-10 milliseconds
                    // It was mostly 0 milliseconds in most cases

                } else if (input == 3) {
                    System.out.println("View the Blockchain");
                    System.out.println(blockChain.toString());
                    break;
                } else if (input == 4) {
                    System.out.println("Corrupt the Blockchain");
                    System.out.println("Enter block ID of block to
Corrupt:");
                    int blockId = scanner.nextInt();
                    System.out.println("Enter new data for block " +
blockId);

                    //After nextInt only the first word is read, so we need
to use nextLine to read the entire line
                    //Code taken from
https://stackoverflow.com/questions/39514730/how-to-take-input-as-string-
with-spaces-in-java-using-scanner
                    scanner.nextLine();
                    String data = scanner.nextLine();
                    Block block = blockChain.getBlock(blockId);
                    block.setData(data);
                    System.out.println("Block " + blockId + " now holds " +
data);
                    break;
                } else if (input == 5) {
                    System.out.println("Repairing the entire chain");
                     startTime = System.currentTimeMillis();
                     blockChain.repairChain();
                     endTime = System.currentTimeMillis();
                    elapsedTimeInMilliSeconds = (endTime - startTime);
                    System.out.println("Total execution time required to
repair the chain was " + elapsedTimeInMilliSeconds + "milliseconds");
                    break;

                    // Time required to repair the chain increased with
increasing difficulty of the most recent block.
                    // It increases exponentially as the difficulty
increases.
                    // For instance:
                    // Time required to repair the chain with the most recent
block of difficulty 2 was 7 milliseconds
                    // Time required to repair the chain with the most recent
block of difficulty 3 was 18 milliseconds
```

```
                    // Time required to repair the chain with the most recent
block of difficulty 4 was 27 milliseconds
                    // Time required to repair the chain with the most recent
block of difficulty 5 was 1443 milliseconds
                    //Time required to repair the chain with the most recent
block of difficulty 6 was 8712 milliseconds

                } else if (input == 6) {
                    System.exit(0);
                }

            }
        }
    }
}
```

Task 0 Hash.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from Lab 1: https://github.com/CMU-Heinz-95702/Lab1-
InstallationAndRaft
// Project 3 Task 0

package blockchaintask0;
import java.security.MessageDigest;
public class Hash {

    public static String hash(String s) throws Exception {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(s.getBytes());
        return bytesToHex(md.digest());
    }

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

Project3Task1
Task 1 Client Side Execution

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ
IDEA.app/Contents/lib/idea_rt.jar=51452:/Applications/IntelliJ IDEA.app/Contents/bin -
Dfile.encoding=UTF-8 -classpath
/Users/gupta/DSProjects/Project3Task1/target/classes:/Users/gupta/.m2/repository/com
/google/code/gson/gson/2.9.0/gson-2.9.0.jar blockchaintask1.BlockChainClient
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.
Enter a number between 0 and 6:
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: 2554278
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 3
Chain hash:
005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F67717584A48EB7939A21

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.
Enter a number between 0 and 6:
1
Enter difficulty > 0:
2
Enter transaction:
Mike pays Marty 100 DSCoin
```

```
Total execution time to add this block was 39.0 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.
Enter a number between 0 and 6:
1
Enter difficulty > 0:
2
Enter transaction:
Marty pays Joe 50 DSCoin
Total execution time to add this block was 12.0 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.
Enter a number between 0 and 6:
1
Enter difficulty > 0:
2
Enter transaction:
Joe pays Andy 10 DS Coin
Total execution time to add this block was 9.0 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.
Enter a number between 0 and 6:
2
Chain Verification :TRUE
```

```
Total execution time to verify the chain was 8.0 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.
Enter a number between 0 and 6:
3
View the Blockchain
{"ds_chain" : [{"index":0,"timestamp":"Oct 27, 2023, 10:21:35
PM","Tx":"Genesis","PrevHash":"","nonce":3,"difficulty":2},
{"index":1,"timestamp":"Oct 27, 2023, 10:21:57 PM","Tx":"Mike pays Marty 100
DSCoin","PrevHash":"005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F67717584A
48EB7939A21","nonce":118,"difficulty":2},
{"index":2,"timestamp":"Oct 27, 2023, 10:22:08 PM","Tx":"Marty pays Joe 50
DSCoin","PrevHash":"00F6ACC3ACDE58F959A9DC8354E7544C44BF55DA198FFF20B0F05
D3CBBC8481A","nonce":185,"difficulty":2},
{"index":3,"timestamp":"Oct 27, 2023, 10:22:17 PM","Tx":"Joe pays Andy 10 DS
Coin","PrevHash":"00BA7B51131651DF02D4414602C952E0498262772D85F8A68BCF8CD
A680A93EE","nonce":15,"difficulty":2}
 ], "chainHash" :
"00EB6048EF64939EFC66574A075D07751E6C1A473CEE09681DBA016924F1F77E"}
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.
Enter a number between 0 and 6:
4
Corrupt the Blockchain
Enter block ID to corrupt:
1
Enter new data for the block:
Mike pays Marty 76 DSCoin
Block 1 now holds Mike pays Marty 76 DSCoin
0. View basic blockchain status.
```

```
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit.
Enter a number between 0 and 6:
3
View the Blockchain
{"ds_chain" : [{"index":0,"timestamp":"Oct 27, 2023, 10:21:35
PM","Tx":"Genesis","PrevHash":"","nonce":3,"difficulty":2},
{"index":1,"timestamp":"Oct 27, 2023, 10:21:57 PM","Tx":"Mike pays Marty 76
DSCoin","PrevHash":"005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F67717584A
48EB7939A21","nonce":118,"difficulty":2},
{"index":2,"timestamp":"Oct 27, 2023, 10:22:08 PM","Tx":"Marty pays Joe 50
DSCoin","PrevHash":"00F6ACC3ACDE58F959A9DC8354E7544C44BF55DA198FFF20B0F05
D3CBBC8481A","nonce":185,"difficulty":2},
{"index":3,"timestamp":"Oct 27, 2023, 10:22:17 PM","Tx":"Joe pays Andy 10 DS
Coin","PrevHash":"00BA7B51131651DF02D4414602C952E0498262772D85F8A68BCF8CD
A680A93EE","nonce":15,"difficulty":2}
 ], "chainHash" :
"00EB6048EF64939EFC66574A075D07751E6C1A473CEE09681DBA016924F1F77E"}
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.
Enter a number between 0 and 6:
2
Chain Verification :FALSE

Total execution time to verify the chain was 11.0 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.
Enter a number between 0 and 6:
5
Total execution time to repair the chain was 15.0 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.
Enter a number between 0 and 6:
2
Chain Verification :TRUE

Total execution time to verify the chain was 4.0 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.
Enter a number between 0 and 6:
1
Enter difficulty > 0:
4
Enter transaction:
Andy pays Sean 25 DSCoin
Total execution time to add this block was 108.0 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.
Enter a number between 0 and 6:
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: 2554278
Expected total hashes required for the whole chain: 66560.0
Nonce for most recent block: 42278
Chain hash:
0000A3D9DC981430EC7399137B9E0F6ED4E753690176E2BE83EB86DF933BF8AB

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.
Enter a number between 0 and 6:
1
Enter difficulty > 0:
5
Enter transaction:
Aditi pays Marty 100 DSCoin
Total execution time to add this block was 500.0 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.
Enter a number between 0 and 6:
6

Process finished with exit code 0
```

Task 1 Server Side Execution

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java -
javaagent:/Applications/IntelliJ
IDEA.app/Contents/lib/idea_rt.jar=51449:/Applications/IntelliJ IDEA.app/Contents/bin -
Dfile.encoding=UTF-8 -classpath
```

```
/Users/gupta/DSProjects/Project3Task1/target/classes:/Users/gupta/.m2/repository/com
/google/code/gson/gson/2.9.0/gson-2.9.0.jar blockchaintask1.BlockChainServer
Blockchain server running
We have a visitor
JSON request message from the client: {"action":"View basic blockchain
status.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Viewed basic
blockchain status.","data":"Current size of chain: 1\nDifficulty of most recent block:
2\nTotal difficulty for all blocks: 2\nApproximate hashes per second on this machine:
2554278\nExpected total hashes required for the whole chain: 256.0\nNonce for most
recent block: 3\nChain hash:
005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F67717584A48EB7939A21\n"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Add a transaction to the
blockchain.","data":"Mike pays Marty 100 DSCoin","difficulty":2}
JSON response message to the client: {"status":"SUCCESS","message":"Added a
transaction to the blockchain.","data":"{\"index\":1,\"timestamp\":\"Oct 27, 2023,
10:21:57 PM\",\"Tx\":\"Mike pays Marty 100
DSCoin\",\"PrevHash\":\"005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F677175
84A48EB7939A21\",\"nonce\":118,\"difficulty\":2}"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Add a transaction to the
blockchain.","data":"Marty pays Joe 50 DSCoin","difficulty":2}
JSON response message to the client: {"status":"SUCCESS","message":"Added a
transaction to the blockchain.","data":"{\"index\":2,\"timestamp\":\"Oct 27, 2023,
10:22:08 PM\",\"Tx\":\"Marty pays Joe 50
DSCoin\",\"PrevHash\":\"00F6ACC3ACDE58F959A9DC8354E7544C44BF55DA198FFF20B0
F05D3CBBC8481A\",\"nonce\":185,\"difficulty\":2}"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Add a transaction to the
blockchain.","data":"Joe pays Andy 10 DS Coin","difficulty":2}
JSON response message to the client: {"status":"SUCCESS","message":"Added a
transaction to the blockchain.","data":"{\"index\":3,\"timestamp\":\"Oct 27, 2023,
10:22:17 PM\",\"Tx\":\"Joe pays Andy 10 DS
Coin\",\"PrevHash\":\"00BA7B51131651DF02D4414602C952E0498262772D85F8A68BCF8
CDA680A93EE\",\"nonce\":15,\"difficulty\":2}"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Verify the
blockchain.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Verified the
blockchain.","data":"Chain Verification :TRUE\n"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"View the
blockchain.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Viewed the
blockchain.","data":"{\"ds_chain\" : [{\"index\":0,\"timestamp\":\"Oct 27, 2023, 10:21:35
PM\",\"Tx\":\"Genesis\",\"PrevHash\":\"\",\"nonce\":3,\"difficulty\":2},\n{\"index\":1,\"time
stamp\":\"Oct 27, 2023, 10:21:57 PM\",\"Tx\":\"Mike pays Marty 100
DSCoin\",\"PrevHash\":\"005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F677175
84A48EB7939A21\",\"nonce\":118,\"difficulty\":2},\n{\"index\":2,\"timestamp\":\"Oct 27,
2023, 10:22:08 PM\",\"Tx\":\"Marty pays Joe 50
DSCoin\",\"PrevHash\":\"00F6ACC3ACDE58F959A9DC8354E7544C44BF55DA198FFF20B0
F05D3CBBC8481A\",\"nonce\":185,\"difficulty\":2},\n{\"index\":3,\"timestamp\":\"Oct 27,
2023, 10:22:17 PM\",\"Tx\":\"Joe pays Andy 10 DS
Coin\",\"PrevHash\":\"00BA7B51131651DF02D4414602C952E0498262772D85F8A68BCF8
CDA680A93EE\",\"nonce\":15,\"difficulty\":2}\n ], \"chainHash\" :
\"00EB6048EF64939EFC66574A075D07751E6C1A473CEE09681DBA016924F1F77E\"}"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Corrupt the chain.","data":"Mike pays
Marty 76 DSCoin","difficulty":1}
JSON response message to the client: {"status":"SUCCESS","message":"Corrupted the
chain.","data":"{\"index\":1,\"timestamp\":\"Oct 27, 2023, 10:21:57 PM\",\"Tx\":\"Mike
pays Marty 76
DSCoin\",\"PrevHash\":\"005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F677175
84A48EB7939A21\",\"nonce\":118,\"difficulty\":2}"}
SUCCESS

We have a visitor
```

```
JSON request message from the client: {"action":"View the
blockchain.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Viewed the
blockchain.","data":"{\"ds_chain\" : [{\"index\":0,\"timestamp\":\"Oct 27, 2023, 10:21:35
PM\",\"Tx\":\"Genesis\",\"PrevHash\":\"\",\"nonce\":3,\"difficulty\":2},\n{\"index\":1,\"time
stamp\":\"Oct 27, 2023, 10:21:57 PM\",\"Tx\":\"Mike pays Marty 76
DSCoin\",\"PrevHash\":\"005ADC13DE1C16375C2139C46E68F76B8EA06AFE733F677175
84A48EB7939A21\",\"nonce\":118,\"difficulty\":2},\n{\"index\":2,\"timestamp\":\"Oct 27,
2023, 10:22:08 PM\",\"Tx\":\"Marty pays Joe 50
DSCoin\",\"PrevHash\":\"00F6ACC3ACDE58F959A9DC8354E7544C44BF55DA198FFF20B0
F05D3CBBC8481A\",\"nonce\":185,\"difficulty\":2},\n{\"index\":3,\"timestamp\":\"Oct 27,
2023, 10:22:17 PM\",\"Tx\":\"Joe pays Andy 10 DS
Coin\",\"PrevHash\":\"00BA7B51131651DF02D4414602C952E0498262772D85F8A68BCF8
CDA680A93EE\",\"nonce\":15,\"difficulty\":2}\n ], \"chainHash\" :
\"00EB6048EF64939EFC66574A075D07751E6C1A473CEE09681DBA016924F1F77E\"}"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Verify the
blockchain.","data":"","difficulty":0}
Improper hash on node 1 Does not begin with 00
JSON response message to the client: {"status":"SUCCESS","message":"Verified the
blockchain.","data":"Chain Verification :FALSE\n"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Hide the corruption by repairing the
chain.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Hid the
corruption by repairing the chain.","data":""}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Verify the
blockchain.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Verified the
blockchain.","data":"Chain Verification :TRUE\n"}
SUCCESS

We have a visitor
```

```
JSON request message from the client: {"action":"Add a transaction to the blockchain.","data":"Andy pays Sean 25 DSCoin","difficulty":4}
JSON response message to the client: {"status":"SUCCESS","message":"Added a transaction to the blockchain.","data":"{\"index\":4,\"timestamp\":\"Oct 27, 2023, 10:24:10 PM\",\"Tx\":\"Andy pays Sean 25 DSCoin\",\"PrevHash\":\"002A3FC65327E3DD47D18DCB16D8463AF6117B935D6551B23BF6B0C5A59E982F\",\"nonce\":42278,\"difficulty\":4}"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"View basic blockchain status.","data":"","difficulty":0}
JSON response message to the client: {"status":"SUCCESS","message":"Viewed basic blockchain status.","data":"Current size of chain: 5\nDifficulty of most recent block: 4\nTotal difficulty for all blocks: 12\nApproximate hashes per second on this machine: 2554278\nExpected total hashes required for the whole chain: 66560.0\nNonce for most recent block: 42278\nChain hash: 0000A3D9DC981430EC7399137B9E0F6ED4E753690176E2BE83EB86DF933BF8AB\n"}
SUCCESS

We have a visitor
JSON request message from the client: {"action":"Add a transaction to the blockchain.","data":"Aditi pays Marty 100 DSCoin","difficulty":5}
JSON response message to the client: {"status":"SUCCESS","message":"Added a transaction to the blockchain.","data":"{\"index\":5,\"timestamp\":\"Oct 27, 2023, 10:24:54 PM\",\"Tx\":\"Aditi pays Marty 100 DSCoin\",\"PrevHash\":\"0000A3D9DC981430EC7399137B9E0F6ED4E753690176E2BE83EB86DF933BF8AB\",\"nonce\":413092,\"difficulty\":5}"}
SUCCESS
```

Task 1 Client Source Code

BlockChainClient.java

```java
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://www.andrew.cmu.edu/course/95-702/examples/javadoc/blockchaintask0/BlockChain.html
// Taken code from Project 2 Task 4 - https://github.com/CMU-Heinz-95702/Project-2-Client-Server
// Project 3 Task 1

package blockchaintask1;

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

public class BlockChainClient {
    // Socket related attributes
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;

    /**
     * Initializes the client connection to a server.
     *
     * @param ip   The IP address of the server.
     * @param port The port number to connect to.
     * @throws IOException If a network error occurs.
     */
    public void startConnection(String ip, int port) throws IOException {
        clientSocket = new Socket(ip, port);
        out = new PrintWriter(clientSocket.getOutputStream(), true);
        in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
    }

    /**
     * Sends a request message to the server and awaits a response.
     *
     * @param msg The request message to be sent.
     * @return A ResponseMessage received from the server.
     * @throws IOException If a network error occurs.
     */

    public ResponseMessage sendMessage(RequestMessage msg) throws IOException
{
        out.println(msg.toJson());
        return ResponseMessage.fromJson(in.readLine());
    }

    /**
     * Closes the connection and associated resources.
     *
     * @throws IOException If a network error occurs.
     */
    public void stopConnection() throws IOException {
        in.close();
        out.close();
        clientSocket.close();
    }

    public static void main(String[] args) throws Exception {
        Long startTime, endTime;
        double elapsedTimeInMilliSeconds;

        // Initialize the client and connect to the server
        BlockChainClient client = new BlockChainClient();
        client.startConnection("localhost", 6666);
        Scanner scanner = new Scanner(System.in);

        // Continuous loop to get user input until exit option is chosen
        while (true) {
            // Display available options to user
            System.out.println("0. View basic blockchain status.");
            System.out.println("1. Add a transaction to the blockchain.");
            System.out.println("2. Verify the blockchain.");
            System.out.println("3. View the blockchain.");
            System.out.println("4. Corrupt the chain.");
            System.out.println("5. Hide the corruption by repairing the
chain.");
            System.out.println("6. Exit.");
            System.out.println("Enter a number between 0 and 6:");
            int choice = scanner.nextInt();

            ResponseMessage response = null;

            switch (choice) {
                case 0:
                    // Request basic blockchain status
                    response = client.sendMessage(new RequestMessage("View
basic blockchain status.", "", 0));
                    response.getMessage();
                    System.out.println(response.getData());
                    break;
                case 1:
                    // Add a new transaction to the blockchain
                    System.out.println("Enter difficulty > 0:");
                    int difficulty = scanner.nextInt();
                    System.out.println("Enter transaction:");
                    //After nextInt only the first word is read, so we need
to use nextLine to read the entire line
                    //Code taken from
https://stackoverflow.com/questions/39514730/how-to-take-input-as-string-
with-spaces-in-java-using-scanner
                    scanner.nextLine();
                    String transaction = scanner.nextLine();
                    startTime = System.currentTimeMillis();
                    // Send the request to the server
                    client.sendMessage(new RequestMessage("Add a transaction
to the blockchain.", transaction, difficulty));
                    endTime = System.currentTimeMillis();
                    elapsedTimeInMilliSeconds = (endTime - startTime);
                    System.out.println("Total execution time to add this
block was " + elapsedTimeInMilliSeconds + " milliseconds");
                    break;
                case 2:
                    // Verify the blockchain
                    startTime = System.currentTimeMillis();
                    // Send the request to the server
                    response= client.sendMessage(new RequestMessage("Verify
the blockchain.", "", 0));
                    endTime = System.currentTimeMillis();
                    elapsedTimeInMilliSeconds = (endTime - startTime);
                    response.getMessage();
                    System.out.println(response.getData());
                    System.out.println("Total execution time to verify the
chain was " + elapsedTimeInMilliSeconds + " milliseconds");
                    break;
                case 3:
                    // View the blockchain
                    System.out.println("View the Blockchain");
                    // Send the request to the server
                    response = client.sendMessage(new RequestMessage("View
the blockchain.", "", 0));
                    response.getMessage();
                    System.out.println(response.getData());
                    break;
                case 4:
                    // Corrupt the blockchain
                    System.out.println("Corrupt the Blockchain");
                    System.out.println("Enter block ID to corrupt:");
                    int blockId = scanner.nextInt();
                    System.out.println("Enter new data for the block:");
                    //After nextInt only the first word is read, so we need
to use nextLine to read the entire line
                    //Code taken from
https://stackoverflow.com/questions/39514730/how-to-take-input-as-string-
with-spaces-in-java-using-scanner
                    scanner.nextLine();
                    String data = scanner.nextLine();
                    // Send the request to the server
                    client.sendMessage(new RequestMessage("Corrupt the
chain.", data, blockId));
                    System.out.println("Block " + blockId + " now holds " +
data);
                    break;
                case 5:
                    // Repair the blockchain
                    startTime = System.currentTimeMillis();
                    // Send the request to the server
                    client.sendMessage(new RequestMessage("Hide the
corruption by repairing the chain.", "", 0));
                    endTime = System.currentTimeMillis();
                    elapsedTimeInMilliSeconds = (endTime - startTime);
                    System.out.println("Total execution time to repair the
chain was " + elapsedTimeInMilliSeconds + " milliseconds");
                    break;
                case 6:
                    // Close connection and exit
                    client.stopConnection();
                    System.exit(0);
                    break;
            }

        }
    }
}
```

RequestMessage.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3 Prerequisites
// Project 3 Task 1
```

```java
package blockchaintask1;

import com.google.gson.Gson;

/**
 * Represents a request message in the blockchain system.
 * The class provides methods for serialization and deserialization using
JSON.
 */
public class RequestMessage {

    private String action;
    private String data;
    private int difficulty;

    /**
     * Constructs a RequestMessage with the specified action, data, and
difficulty level.
     *
     * @param action     The action associated with the request.
     * @param data       The data content for the request.
     * @param difficulty The difficulty level associated with the request, if
any.
     */
    public RequestMessage(String action, String data, int difficulty) {
        this.action = action;
        this.data = data;
        this.difficulty = difficulty;
    }

    /**
     * Gets the action associated with the request.
     *
     * @return The action of the request.
     */
    public String getAction() {
        return action;
    }

    /**
     * Sets the action associated with the request.
     *
     * @param action The action to be set.
     */
    public void setAction(String action) {
        this.action = action;
    }

    /**
     * Gets the data/content of the request.
     *
     * @return The data of the request.
     */
    public String getData() {
        return data;
    }
    /**
     * Sets the data/content of the request.
     *
     * @param data The data to be set.
     */
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Gets the difficulty level associated with the request.
     *
     * @return The difficulty level of the request.
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Sets the difficulty level associated with the request.
     *
     * @param difficulty The difficulty level to be set.
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Deserializes a JSON string into a RequestMessage object.
     *
     * @param json The JSON string representing a RequestMessage.
     * @return The RequestMessage object.
     */
    public static RequestMessage fromJson(String json) {
        return new Gson().fromJson(json, RequestMessage.class);
    }

    /**
     * Serializes the RequestMessage object into a JSON string.
     *
     * @return The JSON string representation of the RequestMessage.
     */
    public String toJson() {
        return new Gson().toJson(this);
    }
}
```

ResponseMessage.java

```java
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3 Prerequisites
// Project 3 Task 1

package blockchaintask1;

import com.google.gson.Gson;
```

```
/**
 * Represents a response message in the blockchain system.
 * The class provides methods for serialization and deserialization using
JSON.
 */
public class ResponseMessage {

    private String status;
    private String message;
    private String data;

    /**
     * Constructs a ResponseMessage with the specified status, message, and
data.
     *
     * @param status  The status of the response.
     * @param message A brief message describing the response.
     * @param data    The actual data or content of the response.
     */
    public ResponseMessage(String status, String message, String data) {
        this.status = status;
        this.message = message;
        this.data = data;
    }

    /**
     * Gets the status of the response.
     *
     * @return The status of the response.
     */
    public String getStatus() {
        return status;
    }

    /**
     * Sets the status of the response.
     *
     * @param status The status to be set.
     */
    public void setStatus(String status) {
        this.status = status;
    }

    /**
     * Gets the message of the response.
     *
     * @return The message of the response.
     */
    public String getMessage() {
        return message;
    }

    /**
     * Sets the message of the response.
     *
     * @param message The message to be set.
     */
    public void setMessage(String message) {
        this.message = message;
    }

    /**
     * Gets the data/content of the response.
     *
     * @return The data of the response.
     */
    public String getData() {
        return data;
    }

    /**
     * Sets the data/content of the response.
     *
     * @param data The data to be set.
     */
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Deserializes a JSON string into a ResponseMessage object.
     *
     * @param json The JSON string representing a ResponseMessage.
     * @return The ResponseMessage object.
     */
    public static ResponseMessage fromJson(String json) {
        return new Gson().fromJson(json, ResponseMessage.class);
    }

    /**
     * Serializes the ResponseMessage object into a JSON string.
     *
     * @return The JSON string representation of the ResponseMessage.
     */
    public String toJson() {
        return new Gson().toJson(this);
    }
}
```

Task 1 Server Source Code

BlockChainServer.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html
// Taken code from Project 2 Task 4 - https://github.com/CMU-Heinz-
95702/Project-2-Client-Server
// Project 3 Task 1

package blockchaintask1;

import java.net.*;
import java.io.*;

public class BlockChainServer {

    // Server and communication attributes
    private ServerSocket serverSocket;
    private Socket clientSocket;
    private PrintWriter out;
    private BufferedReader in;
    // Blockchain attribute
    private BlockChain blockChain;

    /**
     * Starts the blockchain server on a specified port.
     *
     * @param port The port number to bind the server to.
     * @throws IOException, Exception If a network or other error occurs.
     */
    public void start(int port) throws IOException, Exception {
        System.out.println("Blockchain server running");
        serverSocket = new ServerSocket(port);
        blockChain = new BlockChain();

        // Initialize the blockchain with a genesis block
        Block genesisBlock = new Block(0, blockChain.getTime(), "Genesis",
2);
        blockChain.addBlock(genesisBlock);
        blockChain.computeHashesPerSecond();

        // Continuously listen for client connections
        while (true) {
            clientSocket = serverSocket.accept();
            out = new PrintWriter(clientSocket.getOutputStream(), true);
            in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            String inputLine;
            // Process messages from the client
            while ((inputLine = in.readLine()) != null) {
                System.out.println("We have a visitor");
                RequestMessage request = RequestMessage.fromJson(inputLine);
                System.out.println("JSON request message from the client: " +
inputLine);
                ResponseMessage response = processRequest(request);
                System.out.println("JSON response message to the client: " +
response.toJson());
                System.out.println( response.getStatus());
                out.println(response.toJson());
                System.out.println();
            }

            // Close the client connection
            in.close();
            out.close();
            clientSocket.close();
        }
    }
    /**
     * Processes the client's request and produces a response.
     *
     * @param request The client's request.
     * @return The response to the client's request.
     * @throws Exception If an error occurs.
     */
    private ResponseMessage processRequest(RequestMessage request) throws
Exception {
        ResponseMessage r;
        switch (request.getAction()) {
            // Handle different request actions

            case "View basic blockchain status.":
                StringBuilder basicStatus = new StringBuilder();
                basicStatus.append("Current size of chain: " +
blockChain.blocks.size() + "\n");
                basicStatus.append("Difficulty of most recent block: " +
blockChain.getLatestBlock().getDifficulty() + "\n");
                basicStatus.append("Total difficulty for all blocks: " +
blockChain.getTotalDifficulty() + "\n");
                basicStatus.append("Approximate hashes per second on this
machine: " + blockChain.getHashesPerSecond() + "\n");
                basicStatus.append("Expected total hashes required for the
whole chain: " + blockChain.getTotalExpectedHashes() + "\n");
                basicStatus.append("Nonce for most recent block: " +
blockChain.getLatestBlock().getNonce() + "\n");
                basicStatus.append("Chain hash: " + blockChain.getChainHash()
+ "\n");
                // Return the basic blockchain status
                r = new ResponseMessage("SUCCESS", "Viewed basic blockchain
status.", basicStatus.toString());
                return r;

            // Add a new transaction block to the blockchain
            case "Add a transaction to the blockchain.":
                Block newBlock = new Block(blockChain.blocks.size(),
blockChain.getTime(), request.getData(), request.getDifficulty());
                blockChain.addBlock(newBlock);
                // Return the new block
                r = new ResponseMessage("SUCCESS", "Added a transaction to
the blockchain.", newBlock.toString());
                return r;

            // Verify the integrity of the blockchain
            case "Verify the blockchain.":
                StringBuilder verifyChain = new StringBuilder();
                verifyChain.append("Chain Verification :" +
blockChain.isChainValid() + "\n");
                // Return the blockchain verification status
                r= new ResponseMessage("SUCCESS", "Verified the blockchain.",
verifyChain.toString());
                return r;

            // View the entire blockchain
            case "View the blockchain.":
                // Return the blockchain
                r= new ResponseMessage("SUCCESS", "Viewed the blockchain.",
blockChain.toString());
                return r;

            // Corrupt a specified block in the blockchain
            case "Corrupt the chain.":
                Block block = blockChain.getBlock(request.getDifficulty());
// using difficulty as blockID here for simplicity
                block.setData(request.getData());
                // Return the corrupted block
                r = new ResponseMessage("SUCCESS", "Corrupted the chain.",
block.toString());
                return r;

            // Repair the blockchain to fix any corruptions
            case "Hide the corruption by repairing the chain.":
                blockChain.repairChain();
                // Return the repaired blockchain
                r = new ResponseMessage("SUCCESS", "Hid the corruption by
repairing the chain.", "");
                return r;

            // Default case when the action is not recognized
            default:
                r = new ResponseMessage("ERROR", "Invalid action specified",
"");
                System.exit(0);
                stop();
                return r;
        }
    }

    /**
     * Stops the server and releases resources.
     *
     * @throws IOException If a network error occurs.
     */
    public void stop() throws IOException {
        in.close();
        out.close();
        clientSocket.close();
        serverSocket.close();
    }

    public static void main(String[] args) throws Exception {
        // Start the blockchain server
        BlockChainServer server = new BlockChainServer();
        server.start(6666);

    }
}
```

RequestMessage.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3 Prerequisites
// Project 3 Task 1

package blockchaintask1;

import com.google.gson.Gson;

/**
 * Represents a request message in the blockchain system.
 * The class provides methods for serialization and deserialization using
JSON.
 */
public class RequestMessage {

    private String action;
    private String data;
    private int difficulty;

    /**
     * Constructs a RequestMessage with the specified action, data, and
difficulty level.
     *
     * @param action     The action associated with the request.
     * @param data       The data content for the request.
     * @param difficulty The difficulty level associated with the request, if
any.
     */
    public RequestMessage(String action, String data, int difficulty) {
        this.action = action;
        this.data = data;
        this.difficulty = difficulty;
    }

    /**
     * Gets the action associated with the request.
     *
     * @return The action of the request.
     */
    public String getAction() {
        return action;
    }

    /**
     * Sets the action associated with the request.
     *
     * @param action The action to be set.
     */
    public void setAction(String action) {
        this.action = action;
    }

    /**
     * Gets the data/content of the request.
     *
```

```java
     * @return The data of the request.
     */
    public String getData() {
        return data;
    }

    /**
     * Sets the data/content of the request.
     *
     * @param data The data to be set.
     */
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Gets the difficulty level associated with the request.
     *
     * @return The difficulty level of the request.
     */
    public int getDifficulty() {
        return difficulty;
    }

    /**
     * Sets the difficulty level associated with the request.
     *
     * @param difficulty The difficulty level to be set.
     */
    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    /**
     * Deserializes a JSON string into a RequestMessage object.
     *
     * @param json The JSON string representing a RequestMessage.
     * @return The RequestMessage object.
     */
    public static RequestMessage fromJson(String json) {
        return new Gson().fromJson(json, RequestMessage.class);
    }

    /**
     * Serializes the RequestMessage object into a JSON string.
     *
     * @return The JSON string representation of the RequestMessage.
     */
    public String toJson() {
        return new Gson().toJson(this);
    }
}
```

ResponseMessage.java

```java
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3 Prerequisites
// Project 3 Task 1

package blockchaintask1;

import com.google.gson.Gson;

/**
 * Represents a response message in the blockchain system.
 * The class provides methods for serialization and deserialization using
JSON.
 */
public class ResponseMessage {

    private String status;
    private String message;
    private String data;

    /**
     * Constructs a ResponseMessage with the specified status, message, and
data.
     *
     * @param status  The status of the response.
     * @param message A brief message describing the response.
     * @param data    The actual data or content of the response.
     */
    public ResponseMessage(String status, String message, String data) {
        this.status = status;
        this.message = message;
        this.data = data;
    }

    /**
     * Gets the status of the response.
     *
     * @return The status of the response.
     */
    public String getStatus() {
        return status;
    }

    /**
     * Sets the status of the response.
     *
     * @param status The status to be set.
     */
    public void setStatus(String status) {
        this.status = status;
    }

    /**
     * Gets the message of the response.
     *
     * @return The message of the response.
     */
    public String getMessage() {
        return message;
    }

    /**
     * Sets the message of the response.
     *
     * @param message The message to be set.
     */
    public void setMessage(String message) {
        this.message = message;
    }

    /**
     * Gets the data/content of the response.
     *
     * @return The data of the response.
     */
    public String getData() {
        return data;
    }

    /**
     * Sets the data/content of the response.
     *
     * @param data The data to be set.
     */
    public void setData(String data) {
        this.data = data;
    }

    /**
     * Deserializes a JSON string into a ResponseMessage object.
     *
     * @param json The JSON string representing a ResponseMessage.
     * @return The ResponseMessage object.
     */
    public static ResponseMessage fromJson(String json) {
        return new Gson().fromJson(json, ResponseMessage.class);
    }

    /**
     * Serializes the ResponseMessage object into a JSON string.
     *
     * @return The JSON string representation of the ResponseMessage.
     */
    public String toJson() {
        return new Gson().toJson(this);
    }
}
```

Task 1 Block.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/Block.html
// Taken code from https://www.baeldung.com/java-blockchain
// Project 3 Task 1

package blockchaintask1;

import com.google.gson.Gson;
import com.google.gson.annotations.SerializedName;

import java.math.BigInteger;

/**
 * This class represents a simple Block.
 *
 * Each Block object has an index - the position of the block on the chain.
 * The first block (the so called Genesis block) has an index of 0.
 *
 * Each block has a timestamp - a Java Timestamp object, it holds the time of
the block's creation.
 *
 * Each block has a field named data - a String holding the block's single
transaction details.
 *
 * Each block has a String field named previousHash - the SHA256 hash of a
block's parent.
 * This is also called a hash pointer.
 *
 * Each block holds a nonce - a BigInteger value determined by a proof of
work routine.
 * This has to be found by the proof of work logic.
 * It has to be found so that this block has a hash of the proper difficulty.
 * The difficulty is specified by a small integer representing the minimum
number of leading hex zeroes the hash must have.
 *
 * Each block has a field named difficulty - it is an int that specifies the
minimum number of left most hex digits needed by a proper hash.
 * The hash is represented in hexadecimal. If, for example, the difficulty is
3, the hash must have at least three leading hex 0's (or,1 and 1/2 bytes).
 * Each hex digit represents 4 bits.
 */
public class Block extends java.lang.Object{
    private int index;
    private java.sql.Timestamp timestamp;
    @SerializedName(value = "Tx")
    private java.lang.String data;
    @SerializedName(value = "PrevHash")
    private java.lang.String previousHash;
    private BigInteger nonce = BigInteger.ZERO;
    private int difficulty;

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

    /**
     * This method computes a hash of the concatenation of the index,
timestamp, data,
     * previousHash, nonce, and difficulty.
     *
     * Returns: a String holding Hexadecimal characters
     */

    public java.lang.String calculateHash() throws Exception {
        String total = "";
        total+=String.valueOf(this.index);

        // Code taken from https://www.tutorialspoint.com/java-sql-timestamp-
tostring-method-with-
example#:~:text=The%20toString()%20method%20of,Timestamp%20object%20to%20a%20
String.
        total+=this.timestamp.toString();
        total+=this.data;
        total+=this.previousHash;
        total+=String.valueOf(this.nonce);
        total+=String.valueOf(this.difficulty);
        String hash=Hash.hash(total);
        return hash;
    }

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

    /**
     * The proof of work methods finds a good hash. It increments the nonce
until it produces a good hash.
     *
     * This method calls calculateHash() to compute a hash of the
concatenation of the index, timestamp, data,
     * previousHash, nonce, and difficulty. If the hash has the appropriate
number of leading hex zeroes,
     * it is done and returns that proper hash.
     * If the hash does not have the appropriate number of leading hex
zeroes,
     * it increments the nonce by 1 and tries again. It continues this
process, burning electricity and CPU cycles,
     * until it gets lucky and finds a good hash.
     *
     * Returns:
     * a String with a hash that has the appropriate number of leading hex
zeroes.
     * The difficulty value is already in the block. This is the minimum
number of hex 0's a proper hash must have.
     */

    // Taken  code from https://stackoverflow.com/questions/66649182/not-
able-to-get-4-leading-zeros-in-sha256-hash-proof-of-work-ever-java
    public java.lang.String proofOfWork() throws Exception {
        String hash = "";

        //This line of code suggested by GitHub Copilot
        while(!hash.startsWith("0".repeat(this.difficulty))){
            this.nonce = this.nonce.add(BigInteger.ONE);
            hash = calculateHash();
        }
        return hash;
    }

    /**
     * Simple getter method
     * Returns: difficulty of this block
     */

    public int getDifficulty() {
        return this.difficulty;
    }

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

    /**Override Java's toString method
     * Overrides:
     * toString in class java.lang.Object
     * Returns:
     * A JSON representation of all of this block's data is returned.
     */
    public java.lang.String toString(){
        // Create a Gson object
```

```java
        Gson gson = new Gson();
        // Serialize to JSON
        return gson.toJson(this);
    }


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

    /**Simple getter method

     Returns:
     previous hash
     */
    public java.lang.String getPreviousHash(){
        return this.previousHash;
    }

    /**
     * Simple getter method
     * Returns: index of block
     */
    public int getIndex(){
        return this.index;
    }

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

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

    /**
     * Simple getter method
     * Returns:
     * timestamp of this block
     */
    public java.sql.Timestamp getTimestamp(){
        return this.timestamp;
    }

    /**
     * Simple getter method
     * Returns:
     * this block's transaction
     */
    public java.lang.String getData(){
        return this.data;
    }

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

    public static void main(java.lang.String[] args){
    }

}
```

Task 1 BlockChain.java

```
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html
// Taken code from https://www.baeldung.com/java-blockchain
// Project 3 Task 1

package blockchaintask1;

import com.google.gson.Gson;

import java.util.ArrayList;

public class BlockChain
        extends Object {
     /**
     This BlockChain has exactly three instance members -
      an ArrayList to hold Blocks and a chain hash to hold a SHA256 hash of
the most recently added Block.
      It also maintains an instance variable holding the approximate number
of hashes per second on this computer.
      This constructor creates an empty ArrayList for Block storage.
      This constructor sets the chain hash to the empty string and sets
hashes per second to 0.
*/
     ArrayList<Block> blocks;       // To store blocks
    String chainHash;              // To store the SHA256 hash of the most
recently added block
    long hashesPerSecond;          // To store the approximate number of
hashes per second on this computer
```

```
    public BlockChain() {
        this.blocks = new ArrayList<>();
        this.chainHash = "";
        this.hashesPerSecond = 0;

    }

    /**
     * public java.lang.String getChainHash()
     * Returns:
     * the chain hash.
     */
    public String getChainHash(){
        return chainHash;
    }

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

    /**public blockchaintask1.Block getLatestBlock()
     Returns:
     a reference to the most recently added Block.
     */
    public Block getLatestBlock() {
        return blocks.get(blocks.size() - 1);
    }

    /**
     * computeHashesPerSecond
     * public void computeHashesPerSecond()
     * This method computes exactly 2 million hashes and times how long that
process takes.
     * So, hashes per second is approximated as (2 million / number of
seconds).
     * It is run on start up and sets the instance variable hashesPerSecond.
     * It uses a simple string - "00000000" to hash.
     */
    public void computeHashesPerSecond() throws Exception {
        Long startTime = System.currentTimeMillis();
        int numHash = 0;
        while (numHash < 2000000) {
            Hash.hash("00000000");
            numHash++;
        }
        Long endTime = System.currentTimeMillis();
        this.hashesPerSecond =  (int)(numHash / ((endTime - startTime) /
1000.0));
    }
    /**
     * public int getHashesPerSecond()
     * get hashes per second
     * Returns:
```

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

    /**
     * A new Block is being added to the BlockChain.
     * This new block's previous hash must hold the hash
     * of the most recently added block. After this call on addBlock,
     * the new block becomes the most recently added block on the BlockChain.
     * The SHA256 hash of every block must exhibit proof of work,
     * i.e., have the requisite number of leftmost 0's defined by its
difficulty.
     * Suppose our new block is x. And suppose the old blockchain was
     * a <-- b <-- c <-- d then the chain after addBlock completes is
     * a <-- b <-- c <-- d <-- x. Within the block x, there is a previous
hash field.
     * This previous hash field holds the hash of the block d.
     * The block d is called the parent of x.
     * The block x is the child of the block d.
     * It is important to also maintain a hash of the most recently added
block in a chain hash.
     * Let's look at our two chains again. a <-- b <-- c <-- d.
     * The chain hash will hold the hash of d. After adding x,
     * we have a <-- b <-- c <-- d <-- x. The chain hash now holds the hash
of x.
     * The chain hash is not defined within a block but is defined within the
block chain.
     * The arrows are used to describe these hash pointers.
     * If b contains the hash of a then we write a <-- b.
     * Parameters:
     * newBlock is added to the BlockChain as the most recent block
     */
    public void addBlock(Block newBlock) throws Exception {
        newBlock.setPreviousHash(this.chainHash);
        this.blocks.add(newBlock);
        this.chainHash = newBlock.proofOfWork();
    }

    /**
     * public java.lang.String toString()
     * This method uses the toString method defined on each individual block.
     * Overrides:
     * toString in class java.lang.Object
     * Returns:
     * a String representation of the entire chain is returned.
     */

    /**
     * View the Blockchain
     * {"ds_chain" : [ {"index" : 0,"time stamp " : "2022-02-25
17:41:11.927","Tx ": "Genesis","PrevHash" : "","nonce" : 286,"difficulty":
2},
     * {"index" : 1,"time stamp " : "2022-02-25 17:42:46.053","Tx ": "Mike
pays Marty 100 DSCoin","PrevHash" :
"0026883909AA470264145129F134489316E6A38439240D0468D69AA9665D4993","nonce" :
```

```
165,"difficulty": 2},
     * {"index" : 2,"time stamp " : "2022-02-25 17:44:27.43","Tx ": "Marty
pays Joe 50 DSCoin","PrevHash" :
"000D14B83028DD36BD6330B8DAB185012F8625E9C9D1A8704E9C1189FD98D9DF","nonce" :
819,"difficulty": 2},
     * {"index" : 3,"time stamp " : "2022-02-25 17:45:22.044","Tx ": "Joe
pays Andy 10 DSCoin","PrevHash" :
"00B4CC539C5CC36AE2F09CC7B857A1330D2D02C00CA90D4A34ACD7E01D7225FC","nonce" :
167,"difficulty": 2}
     */
    //override toString method to give above output

    @Override
    public String toString() {

        StringBuilder sb = new StringBuilder();
        sb.append("{\"ds_chain\" : [");

        for (int i = 0; i < blocks.size(); i++) {
            sb.append(blocks.get(i).toString());  // This will serialize each
block to JSON

            if (i != blocks.size() - 1) {  // if not the last block, append a
comma
                sb.append(",");
            }
            //Code from https://stackoverflow.com/questions/14534767/how-to-
append-a-newline-to-stringbuilder - for reference
            sb.append("\n");
        }
        //Code from https://stackoverflow.com/questions/22682016/how-to-use-
string-builder-to-append-double-quotes-and-or-between-elements
        sb.append(" ], \"chainHash\" :
").append("\"").append(this.chainHash).append("\"").append("}");
        return sb.toString();
    }

    /**
     * public blockchaintask1.Block getBlock(int i)
     * return block at position i
     * Parameters:
     * i -
     * Returns:
     * block at postion i
     */
    public Block getBlock(int i){
        return blocks.get(i);
    }

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

    /**
    * public double getTotalExpectedHashes()
    * Compute and return the expected number of hashes required for the
entire chain.
    * Returns:
    * totalExpectedHashes
    */
    public double getTotalExpectedHashes(){
        double totalExpectedHashes = 0;
        for (Block block : blocks) {
            //Taken code from
https://bitcoin.stackexchange.com/questions/4565/calculating-average-number-
of-hashes-tried-before-hitting-a-valid-block
            totalExpectedHashes += Math.pow(16, block.getDifficulty());
        }
        return totalExpectedHashes;
    }

    /**
     * public java.lang.String isChainValid()
     * If the chain only contains one block, the genesis block at position 0,
     * this routine computes the hash of the block and
     * checks that the hash has the requisite number of leftmost 0's (proof
of work) as specified in the difficulty field.
     * It also checks that the chain hash is equal to this computed hash.
     * If either check fails, return an error message. Otherwise, return the
string "TRUE".
     * If the chain has more blocks than one, begin checking from block one.
     * Continue checking until you have validated the entire chain.
     * The first check will involve a computation of a hash in Block 0 and a
comparison with the hash pointer in Block 1.
     * If they match and if the proof of work is correct, go and visit the
next block in the chain.
     * At the end, check that the chain hash is also correct.
     * Returns:
     * "TRUE" if the chain is valid, otherwise return a string with an
appropriate error message
     */

    // https://www.baeldung.com/java-blockchain - for reference
    public String isChainValid() throws Exception {
        if (blocks.size() == 1) {
            Block block = blocks.get(0);
            String hash = block.calculateHash();
            if (hash.startsWith("0".repeat(block.getDifficulty()))) {
                if (hash.equals(this.chainHash)) {
                    return "TRUE";
                } else {
                    System.out.println("Improper hash on node 0");
                    return "FALSE";
                }
            } else {
                System.out.println("Does not begin with " +
("0".repeat(block.getDifficulty())));
                return "FALSE";
            }
        } else {
            for (int i = 1; i < blocks.size(); i++) {
                Block block = blocks.get(i);
                Block prevBlock = blocks.get(i - 1);
                String hash = block.calculateHash();
                if (!hash.startsWith("0".repeat(block.getDifficulty()))
|| !block.getPreviousHash().equals(prevBlock.calculateHash())){
                    System.out.println("Improper hash on node " + i + "
Does not begin with " + ("0".repeat(block.getDifficulty())));
                    return "FALSE";
                }
            }
            return "TRUE";
        }
    }

    /**
     * public void repairChain()
     * This routine repairs the chain.
     * It checks the hashes of each block and ensures that any illegal hashes
are recomputed.
     * After this routine is run, the chain will be valid.
     * The routine does not modify any difficulty values.
     * It computes new proof of work based on the difficulty specified in the
Block.
     */
    public void repairChain() throws Exception {
        String hash = "";
        for (int i = 1; i < this.blocks.size(); i++) {
            Block currentBlock = this.blocks.get(i);
            Block prevBlock = this.blocks.get(i - 1);
            //Setting all the hashes again so that the chain is repaired
            currentBlock.setPreviousHash(prevBlock.calculateHash());

            // Getting the proof of work for the current block
            hash=currentBlock.proofOfWork();
        }
        //Assigning the proof of work of the final block to the chain hash
        this.chainHash = hash;
    }
}
```

Project3Task2

Ethereum block number 0

Request (from IntelliJ)

```
###
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3
// Project 3 Task 2

# curl -X POST -H "Content-Type: application/json" --data
'{"jsonrpc":"2.0","method":"eth_getBlockByNumber","params":["0x0",
true],"id":1}' https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4
POST https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4
Content-Type: application/json

{
  "jsonrpc": "2.0",
  "method": "eth_getBlockByNumber",
  "params": [
    "0x0",
    true
  ],
  "id": 1
}

###
```

Response (from IntelliJ)

```
POST https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4

HTTP/1.1 200 OK
Date: Fri, 27 Oct 2023 03:49:16 GMT
Content-Type: application/json
Content-Length: 464
Connection: keep-alive
Content-Encoding: gzip
Vary: Origin
Vary: Accept-Encoding

{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "difficulty": "0x400000000",
    "extraData":
"0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa",
    "gasLimit": "0x1388",
    "gasUsed": "0x0",
    "hash": "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
    "logsBloom":
"0x000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000",
    "miner": "0x0000000000000000000000000000000000000000",
    "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "nonce": "0x0000000000000042",
    "number": "0x0",
    "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "receiptsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "sha3Uncles":
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "size": "0x21c",
    "stateRoot": "0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544",
    "timestamp": "0x0",
    "totalDifficulty": "0x400000000",
    "transactions": [],
    "transactionsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "uncles": []
  }
}
```

Response file saved.
> 2023-10-26T234916.200.json

Response code: 200 (OK); Time: 229ms (229 ms); Content length: 1472 bytes (1.47 kB)

2023-10-26T234916.200.json

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "difficulty": "0x400000000",
    "extraData":
"0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa",
    "gasLimit": "0x1388",
    "gasUsed": "0x0",
    "hash":
"0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
    "logsBloom":
"0x00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000",
    "miner": "0x0000000000000000000000000000000000000000",
    "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "nonce": "0x0000000000000042",
    "number": "0x0",
    "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "receiptsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "sha3Uncles":
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "size": "0x21c",
    "stateRoot":
"0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544",
    "timestamp": "0x0",
    "totalDifficulty": "0x400000000",
    "transactions": [],
    "transactionsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "uncles": []
  }
}
```

Ethereum block number 1

Request (from IntelliJ)

```
###
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3
// Project 3 Task 2

# curl -X POST -H "Content-Type: application/json" --data
'{"jsonrpc":"2.0","method":"eth_getBlockByNumber","params":["0x5BAD55",
true],"id":1}' https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4
POST https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4
Content-Type: application/json

{
  "jsonrpc": "2.0",
  "method": "eth_getBlockByNumber",
  "params": [
    "0x0",
    true
  ],
  "id": 1
}

###
```

Response (from IntelliJ)

POST https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4

HTTP/1.1 200 OK
Date: Fri, 27 Oct 2023 05:38:16 GMT
Content-Type: application/json

```
Content-Length: 464
Connection: keep-alive
Content-Encoding: gzip
Vary: Origin
Vary: Accept-Encoding

{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "difficulty": "0x400000000",
    "extraData":
"0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa",
    "gasLimit": "0x1388",
    "gasUsed": "0x0",
    "hash": "0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
    "logsBloom":
"0x000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000",
    "miner": "0x0000000000000000000000000000000000000000",
    "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "nonce": "0x0000000000000042",
    "number": "0x0",
    "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "receiptsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "sha3Uncles":
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "size": "0x21c",
    "stateRoot": "0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544",
    "timestamp": "0x0",
    "totalDifficulty": "0x400000000",
    "transactions": [],
    "transactionsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "uncles": []
  }
}
Response file saved.
```

> 2023-10-27T013816.200.json

Response code: 200 (OK); Time: 211ms (211 ms); Content length: 1472 bytes (1.47 kB)

2023-10-27T013816.200.json

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "difficulty": "0x400000000",
    "extraData":
"0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa",
    "gasLimit": "0x1388",
    "gasUsed": "0x0",
    "hash":
"0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
    "logsBloom":
"0x00000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000",
    "miner": "0x0000000000000000000000000000000000000000",
    "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "nonce": "0x0000000000000042",
    "number": "0x0",
    "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "receiptsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "sha3Uncles":
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "size": "0x21c",
    "stateRoot":
"0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544",
    "timestamp": "0x0",
    "totalDifficulty": "0x400000000",
    "transactions": [],
    "transactionsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "uncles": []
  }
}
```

Account balance query
Request (from IntelliJ)

```
###
// Author - Aditi Gupta (argupta@andrew.cmu.edu)
// Taken from https://github.com/CMU-Heinz-95702/Project3
// Project 3 Task 2


# curl -X POST -H "Content-Type: application/json" --data
'{"jsonrpc":"2.0","method":"eth_getBalance","params":["0x742d35Cc6634C0532925
a3b844Bc454e4438f44e", "latest"],"id":1}'
https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4
POST https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4
Content-Type: application/json

{
  "jsonrpc": "2.0",
  "method": "eth_getBalance",
  "params": [
    "0x742d35Cc6634C0532925a3b844Bc454e4438f44e",
    "latest"
  ],
  "id": 1
}

###
```

Response (from IntelliJ)

```
POST https://mainnet.infura.io/v3/5a824b061a6846b5a23360b24ffcc1e4

HTTP/1.1 200 OK
Date: Fri, 27 Oct 2023 05:49:27 GMT
Content-Type: application/json
Content-Length: 58
Connection: keep-alive
Vary: Origin
Vary: Accept-Encoding

{
  "jsonrpc": "2.0",
  "id": 1,
  "result": "0x2284fe985005082ed9ce"
}
Response file saved.
> 2023-10-27T014927.200.json

Response code: 200 (OK); Time: 209ms (209 ms); Content length: 58 bytes (58 B)
```

2023-10-27T014927.200.json

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": "0x2284fe985005082ed9ce"
}
```