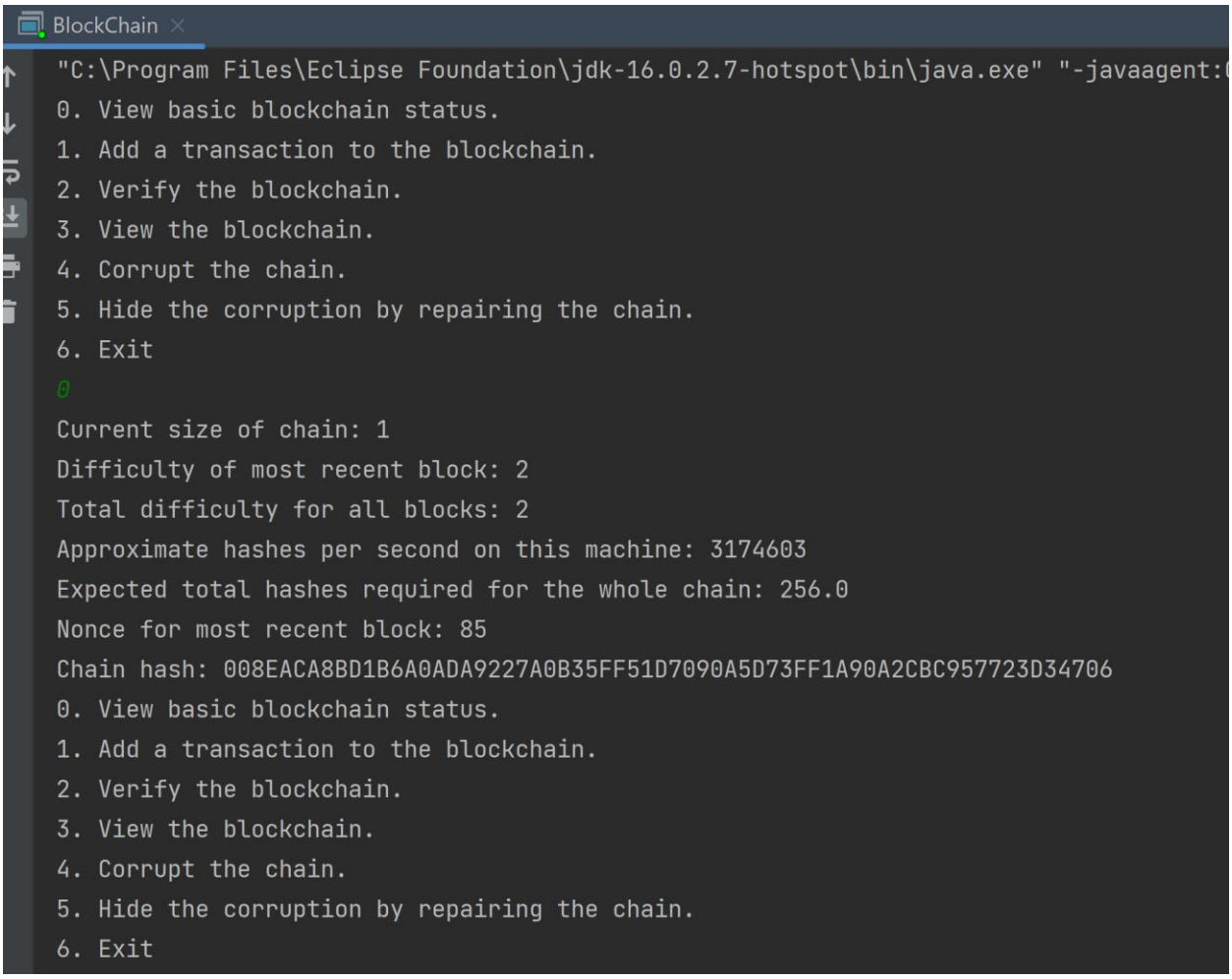
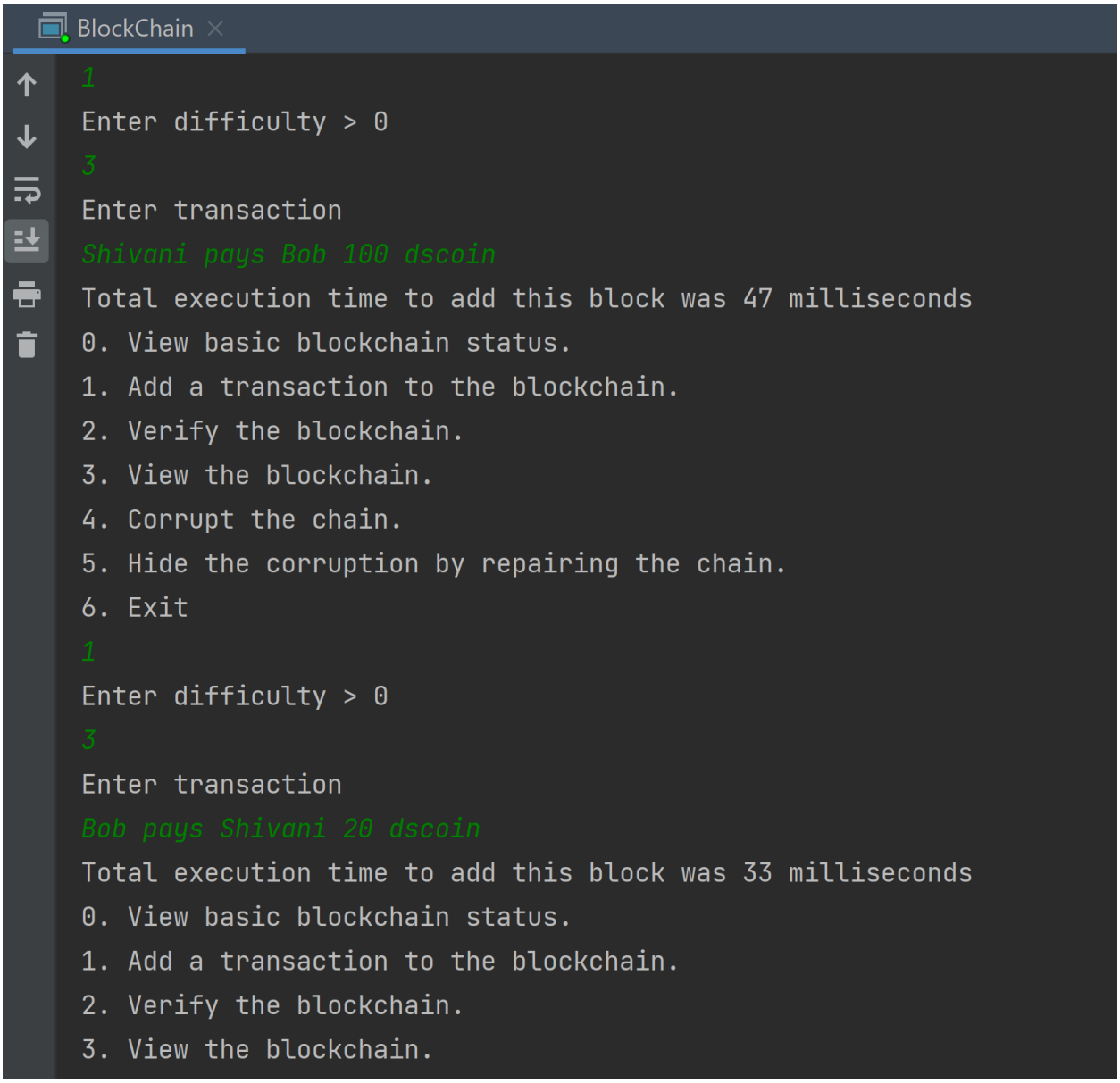
PROJECT 3

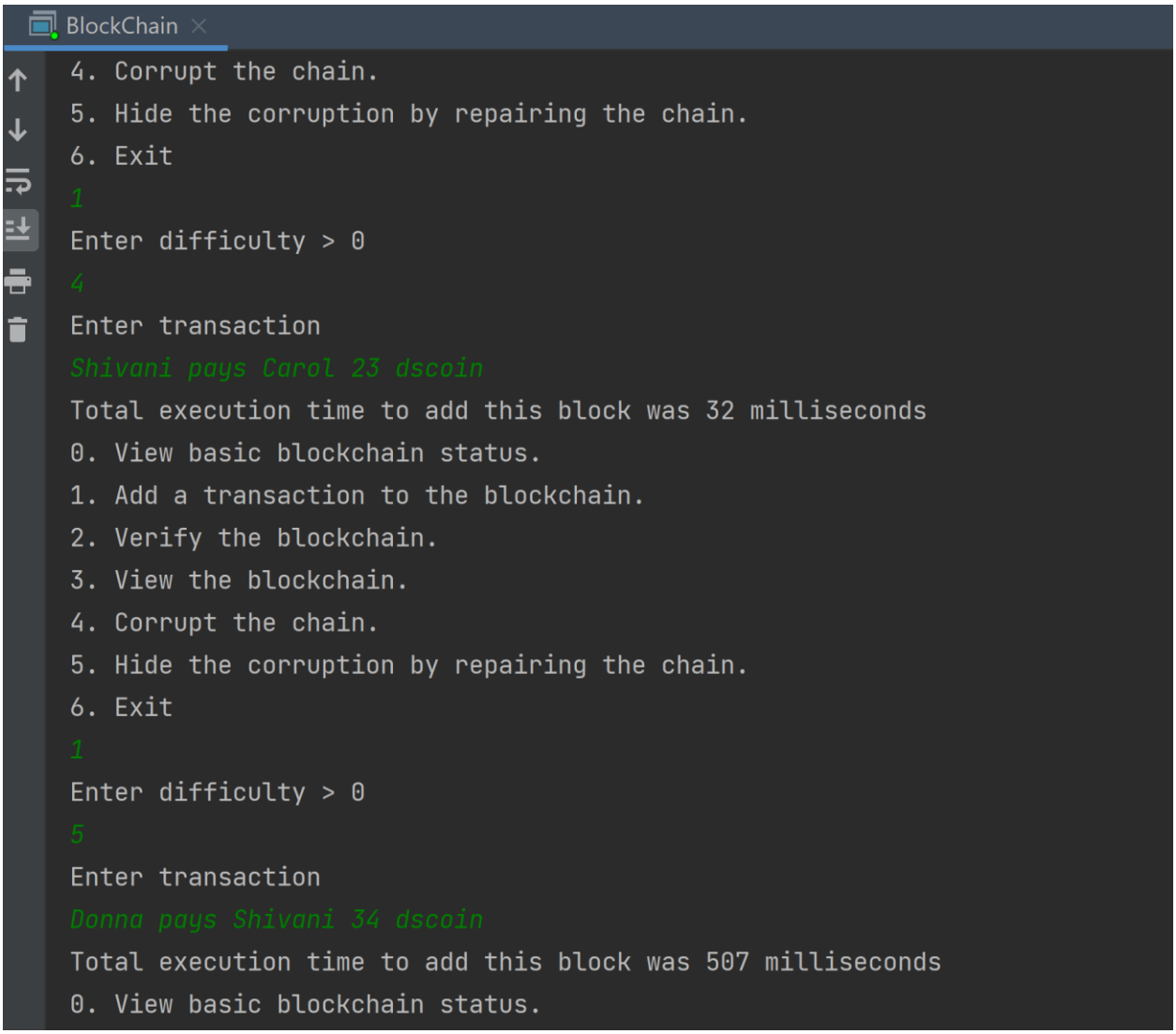
Name: Shivani Poovaiah Ajjikutira

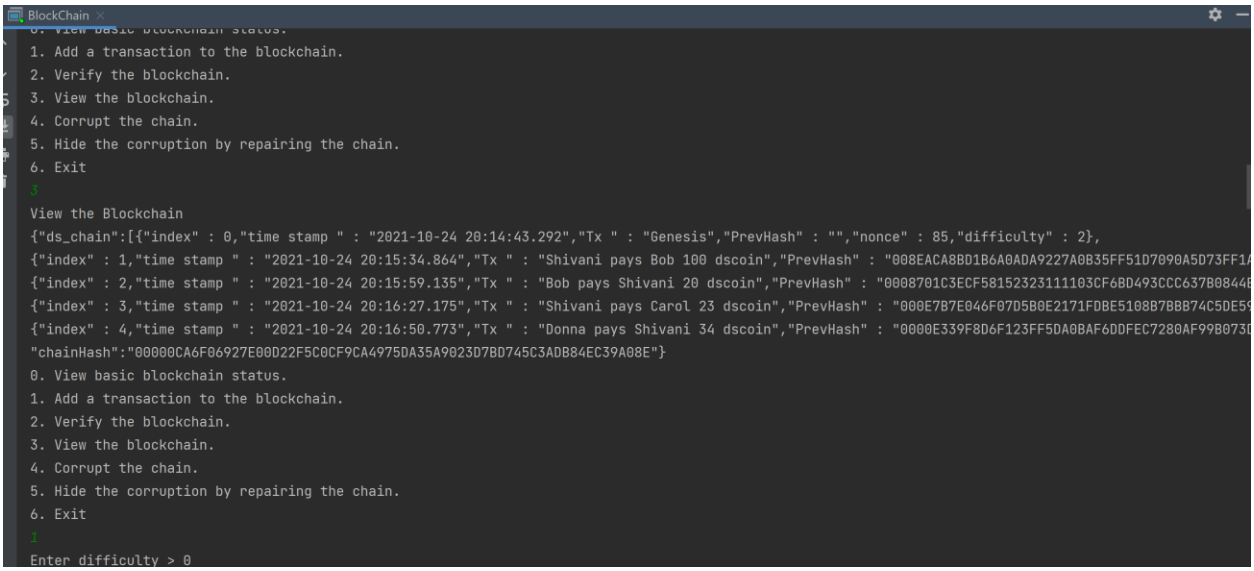
Email ID: sajjikut@andrew.cmu.edu

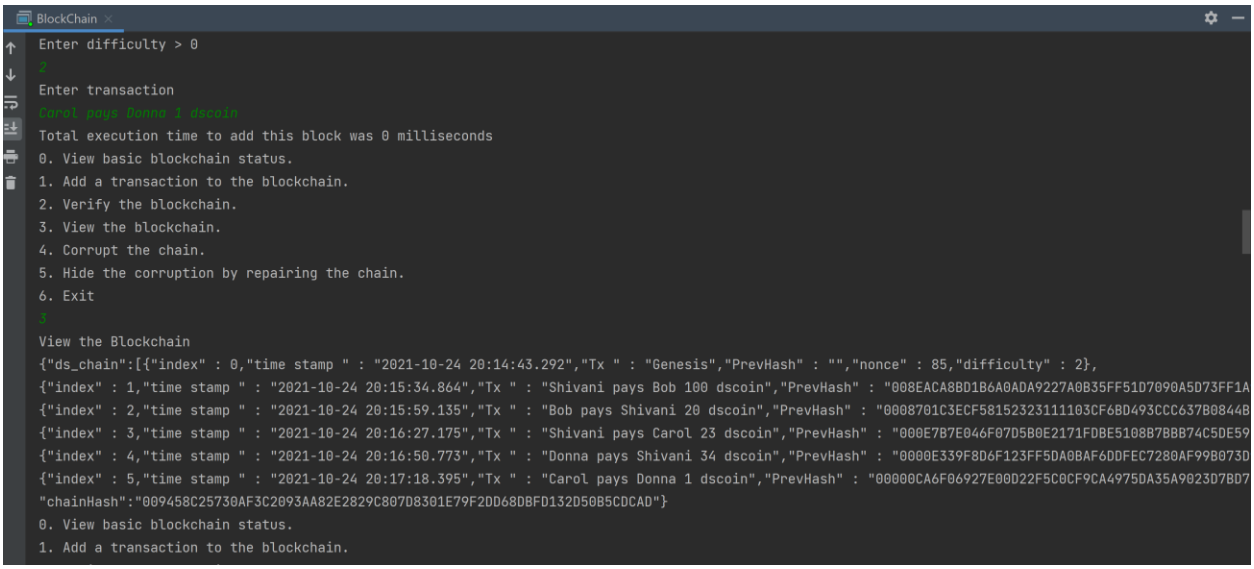
1. Task 0 Execution

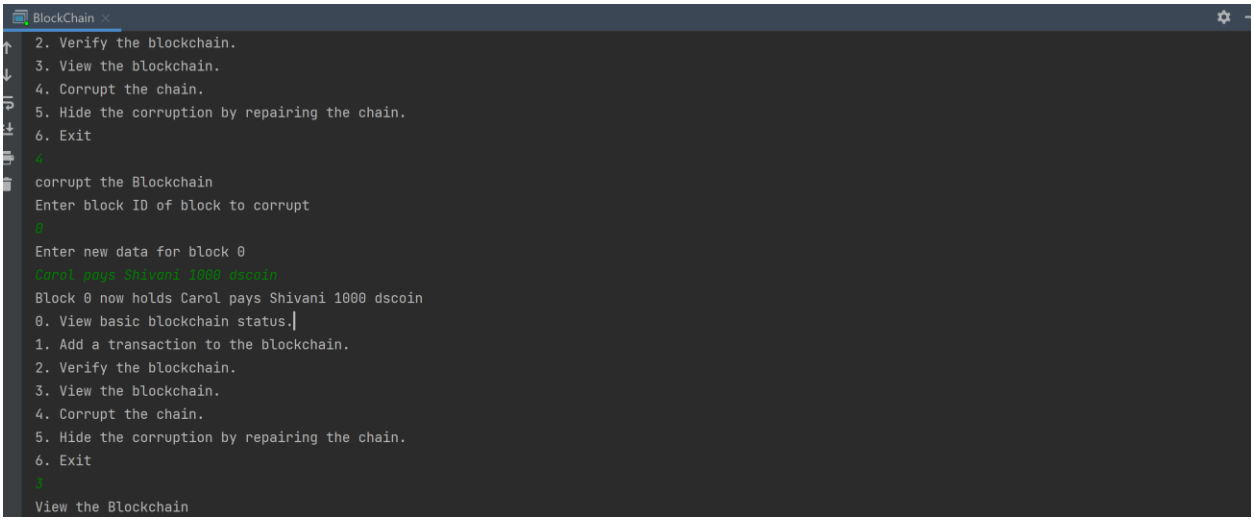


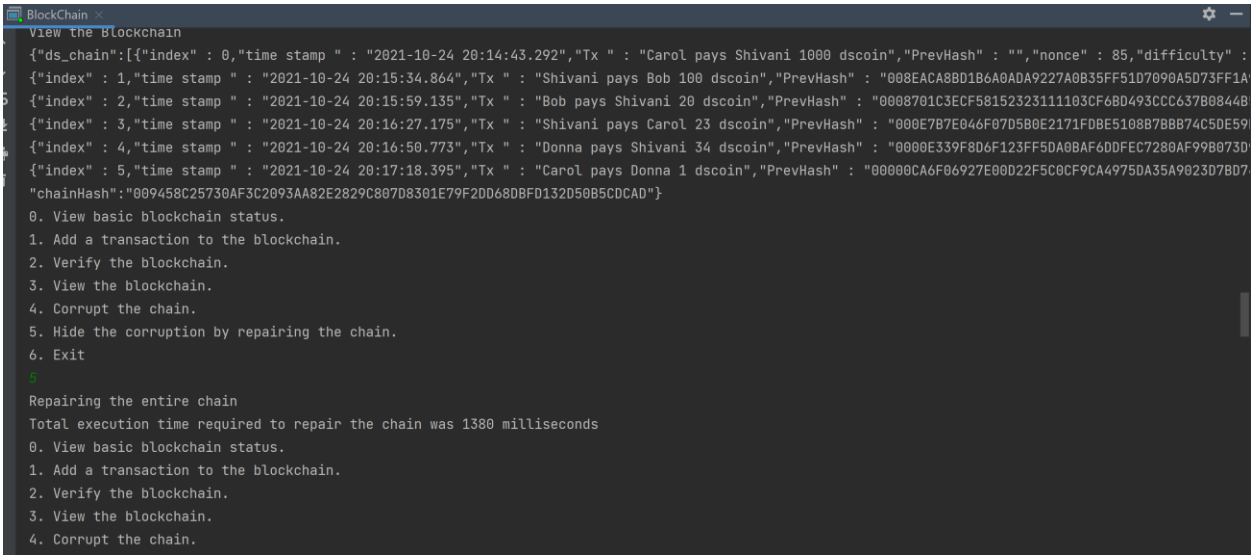

BlockChain
1
Enter difficulty > 0
3
Enter transaction
Shivani pays Bob 100 dscoin
Total execution time to add this block was 47 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
3
Enter transaction
Bob pays Shivani 20 dscoin
Total execution time to add this block was 33 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.

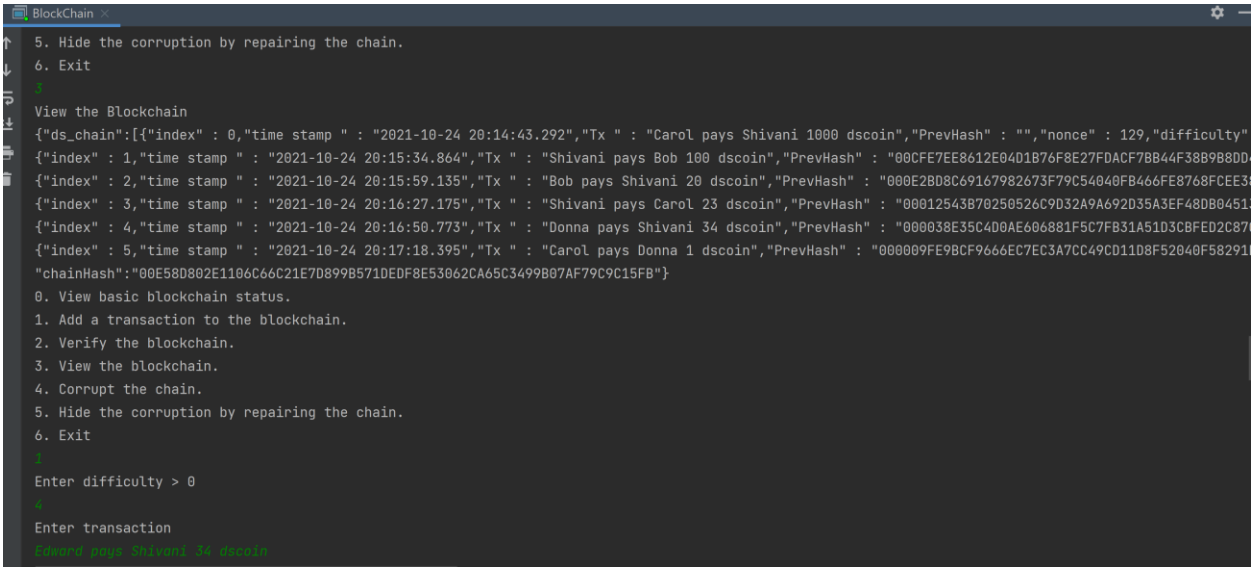

BlockChain
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
4
Enter transaction
Shivani pays Carol 23 dscoin
Total execution time to add this block was 32 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
5
Enter transaction
Donna pays Shivani 34 dscoin
Total execution time to add this block was 507 milliseconds
0. View basic blockchain status.

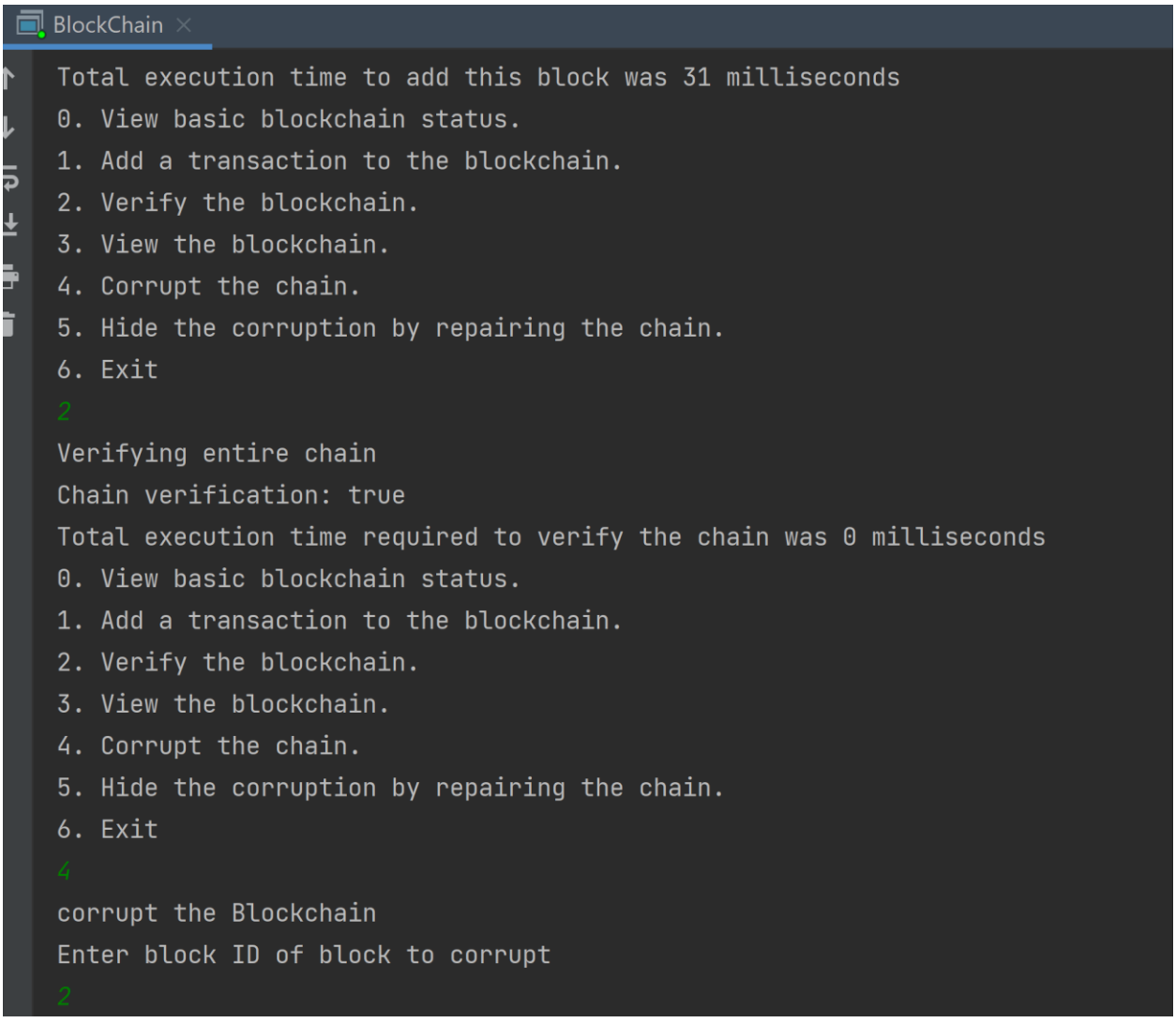

BlockChain
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index" : 0,"time stamp " : "2021-10-24 20:14:43.292","Tx " : "Genesis","PrevHash" : "","nonce" : 85,"difficulty" : 2},
{"index" : 1,"time stamp " : "2021-10-24 20:15:34.864","Tx " : "Shivani pays Bob 100 dscoin","PrevHash" : "008EACA8BD1B6A0ADA9227A0B35FF51D7090A5D73FF1A
{"index" : 2,"time stamp " : "2021-10-24 20:15:59.135","Tx " : "Bob pays Shivani 20 dscoin","PrevHash" : "0008701C3ECF58152323111103CF6BD493CCC637B0844B
{"index" : 3,"time stamp " : "2021-10-24 20:16:27.175","Tx " : "Shivani pays Carol 23 dscoin","PrevHash" : "000E7B7E046F07D5B0E2171FDBE5108B7BBB74C5DE59
{"index" : 4,"time stamp " : "2021-10-24 20:16:50.773","Tx " : "Donna pays Shivani 34 dscoin","PrevHash" : "0000E339F8D6F123FF5DA0BAF6DDFEC7280AF99B073D
"chainHash":"00000CA6F06927E00D22F5C0CF9CA4975DA35A9023D7BD745C3ADB84EC39A08E"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0

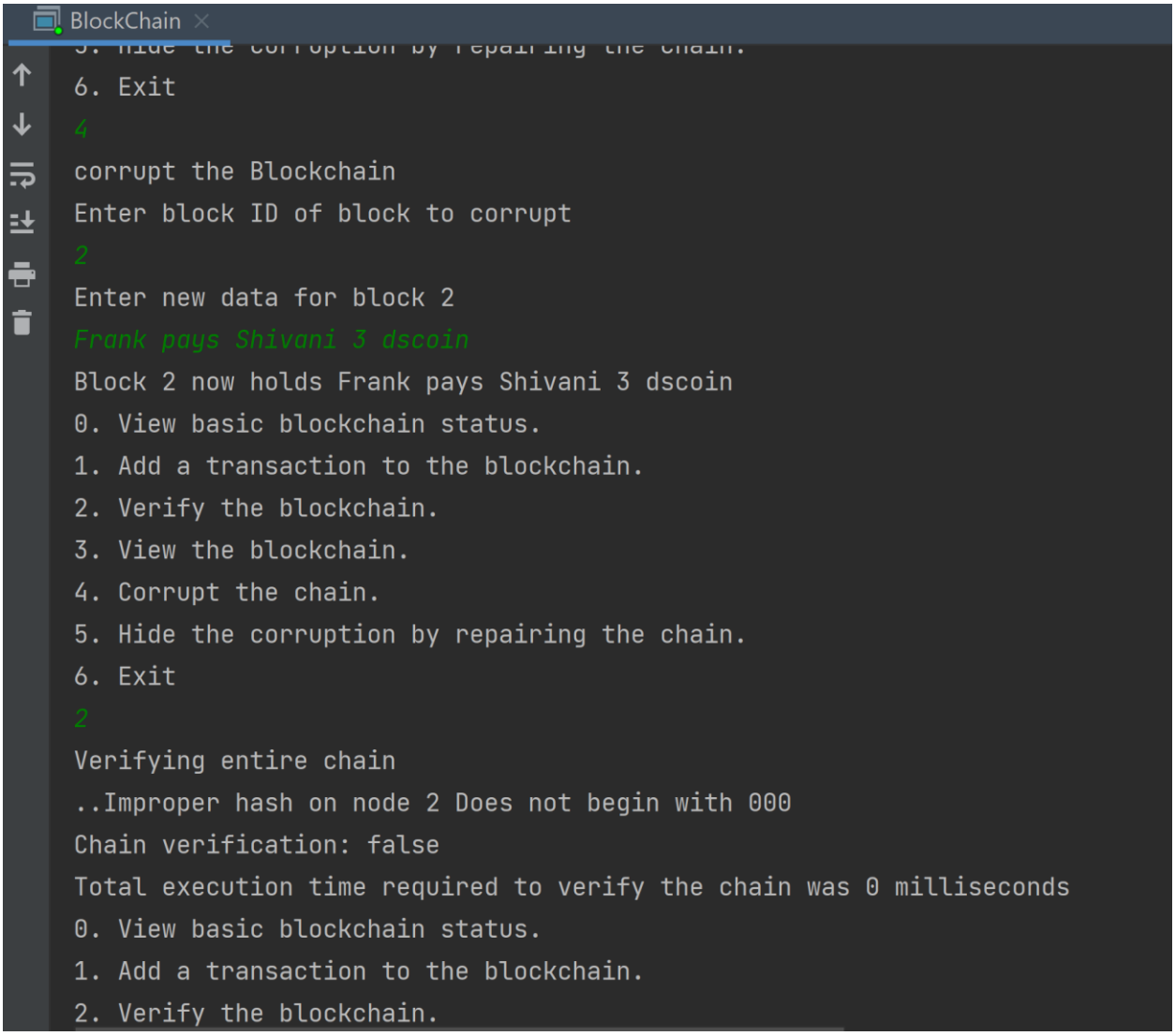

BlockChain
Enter difficulty > 0
2
Enter transaction
Carol pays Donna 1 dscoin
Total execution time to add this block was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index" : 0,"time stamp " : "2021-10-24 20:14:43.292","Tx " : "Genesis","PrevHash" : "","nonce" : 85,"difficulty" : 2},
{"index" : 1,"time stamp " : "2021-10-24 20:15:34.864","Tx " : "Shivani pays Bob 100 dscoin","PrevHash" : "008EACA8BD1B6A0ADA9227A0B35FF51D7090A5D73FF1A
{"index" : 2,"time stamp " : "2021-10-24 20:15:59.135","Tx " : "Bob pays Shivani 20 dscoin","PrevHash" : "0008701C3ECF58152323111103CF6BD493CCC637B0844B
{"index" : 3,"time stamp " : "2021-10-24 20:16:27.175","Tx " : "Shivani pays Carol 23 dscoin","PrevHash" : "000E7B7E046F07D5B0E2171FDBE5108B7BBB74C5DE59
{"index" : 4,"time stamp " : "2021-10-24 20:16:50.773","Tx " : "Donna pays Shivani 34 dscoin","PrevHash" : "0000E339F8D6F123FF5DA0BAF6DDFEC7280AF99B073D
{"index" : 5,"time stamp " : "2021-10-24 20:17:18.395","Tx " : "Carol pays Donna 1 dscoin","PrevHash" : "00000CA6F06927E00D22F5C0CF9CA4975DA35A9023D7BD7
"chainHash":"009458C25730AF3C2093AA82E2829C807D8301E79F2DD68DBFD132D50B5CDCAD"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.


BlockChain
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
4
corrupt the Blockchain
Enter block ID of block to corrupt
0
Enter new data for block 0
Carol pays Shivani 1000 dscoin
Block 0 now holds Carol pays Shivani 1000 dscoin
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain

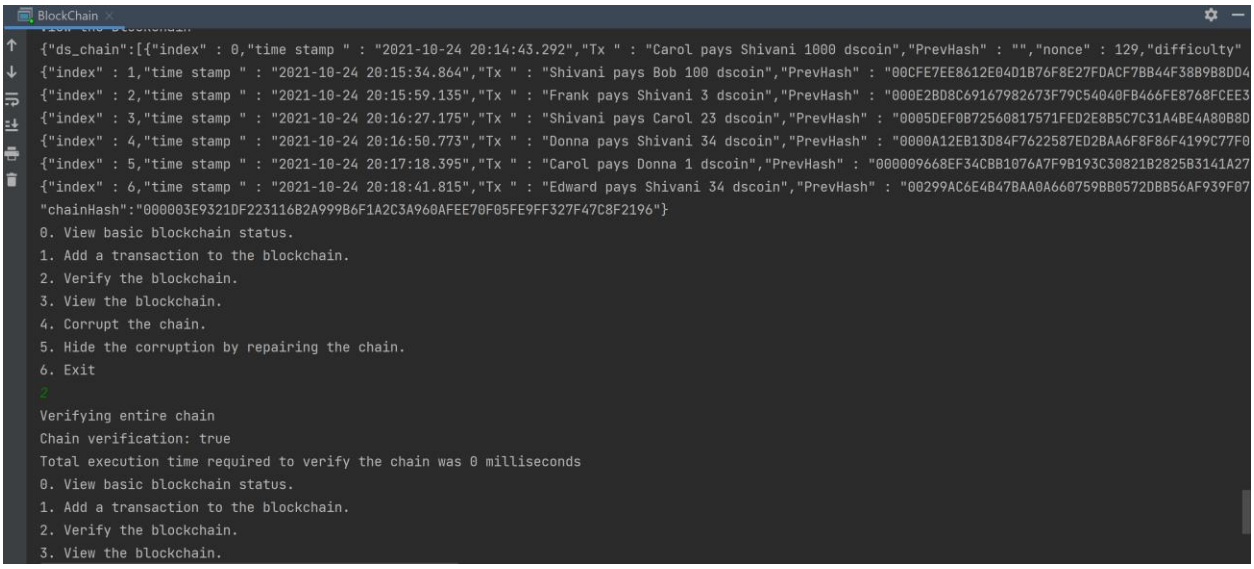

BlockChain
View the Blockchain
{"ds_chain":[{"index" : 0,"time stamp " : "2021-10-24 20:14:43.292","Tx " : "Carol pays Shivani 1000 dscoin","PrevHash" : "","nonce" : 85,"difficulty" :
{"index" : 1,"time stamp " : "2021-10-24 20:15:34.864","Tx " : "Shivani pays Bob 100 dscoin","PrevHash" : "008EACA8BD1B6A0ADA9227A0B35FF51D7090A5D73FF1A
{"index" : 2,"time stamp " : "2021-10-24 20:15:59.135","Tx " : "Bob pays Shivani 20 dscoin","PrevHash" : "0008701C3ECF58152323111103CF6BD493CCC637B0844B
{"index" : 3,"time stamp " : "2021-10-24 20:16:27.175","Tx " : "Shivani pays Carol 23 dscoin","PrevHash" : "000E7B7E046F07D5B0E2171FDBE5108B7BBB74C5DE59
{"index" : 4,"time stamp " : "2021-10-24 20:16:50.773","Tx " : "Donna pays Shivani 34 dscoin","PrevHash" : "0000E339F8D6F123FF5DA0BAF6DDFEC7280AF99B073D
{"index" : 5,"time stamp " : "2021-10-24 20:17:18.395","Tx " : "Carol pays Donna 1 dscoin","PrevHash" : "00000CA6F06927E00D22F5C0CF9CA4975DA35A9023D7BD7
"chainHash":"009458C25730AF3C2093AA82E2829C807D8301E79F2DD68DBFD132D50B5CDCAD"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
5
Repairing the entire chain
Total execution time required to repair the chain was 1380 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.

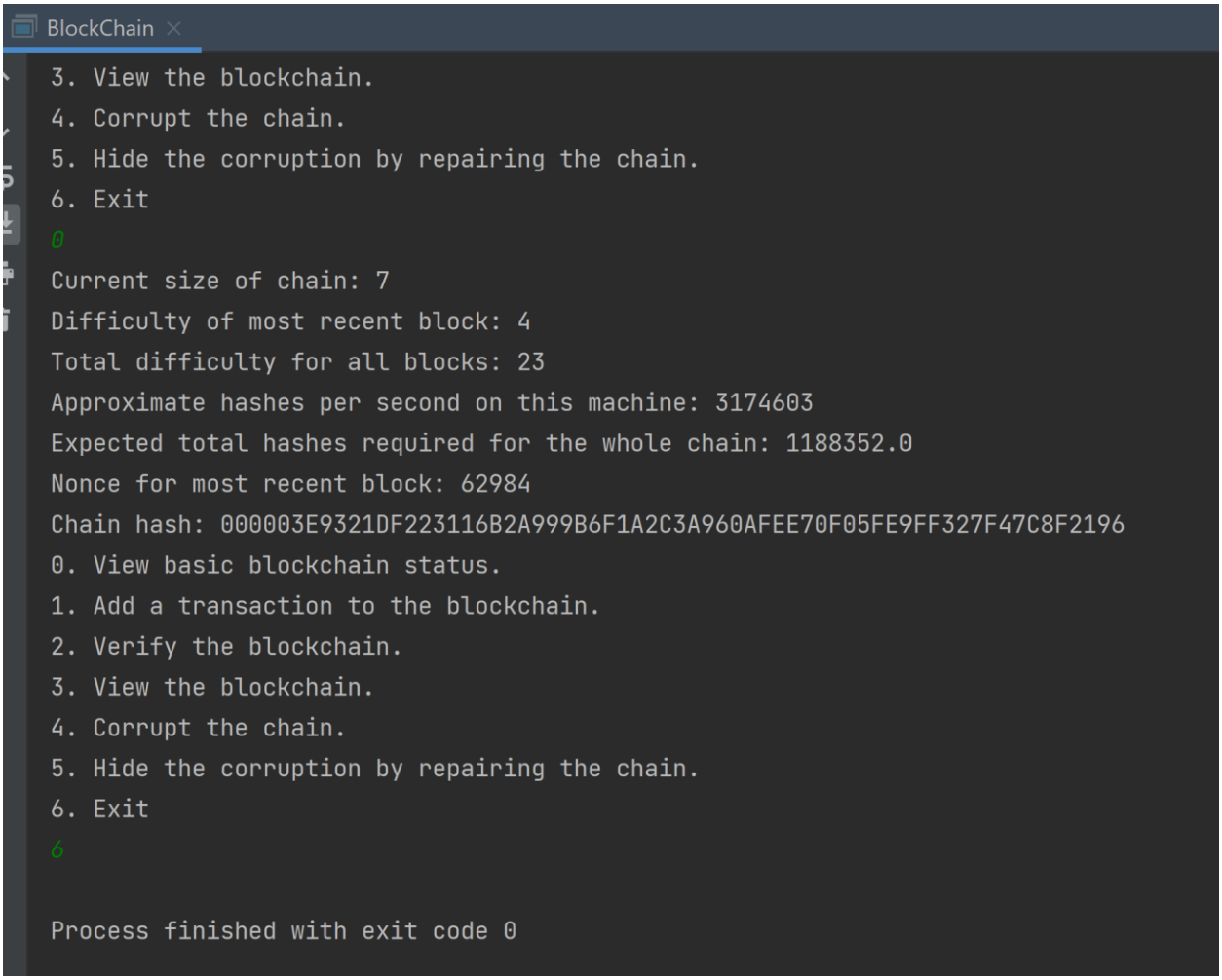

BlockChain
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
{"ds_chain":[{"index" : 0,"time stamp " : "2021-10-24 20:14:43.292","Tx " : "Carol pays Shivani 1000 dscoin","PrevHash" : "","nonce" : 129,"difficulty"
{"index" : 1,"time stamp " : "2021-10-24 20:15:34.864","Tx " : "Shivani pays Bob 100 dscoin","PrevHash" : "00CFE7EE8612E04D1B76F8E27FDACF7BB44F38B9B8DD
{"index" : 2,"time stamp " : "2021-10-24 20:15:59.135","Tx " : "Bob pays Shivani 20 dscoin","PrevHash" : "000E2BD8C69167982673F79C54040FB466FE8768FCEE3
{"index" : 3,"time stamp " : "2021-10-24 20:16:27.175","Tx " : "Shivani pays Carol 23 dscoin","PrevHash" : "00012543B70250526C9D32A9A692D35A3EF48DB0451
{"index" : 4,"time stamp " : "2021-10-24 20:16:50.773","Tx " : "Donna pays Shivani 34 dscoin","PrevHash" : "000038E35C4D0AE606881F5C7FB31A51D3CBFED2C87
{"index" : 5,"time stamp " : "2021-10-24 20:17:18.395","Tx " : "Carol pays Donna 1 dscoin","PrevHash" : "000009FE9BCF9666EC7EC3A7CC49CD11D8F52040F58291
"chainHash":"00E58D802E1106C66C21E7D899B571DEDF8E53062CA65C3499B07AF79C9C15FB"}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
4
Enter transaction
Edward pays Shivani 34 dscoin


BlockChain
Total execution time to add this block was 31 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Verifying entire chain
Chain verification: true
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
4
corrupt the Blockchain
Enter block ID of block to corrupt
2


BlockChain
5. Hide the corruption by repairing the chain.
6. Exit
4
corrupt the Blockchain
Enter block ID of block to corrupt
2
Enter new data for block 2
Frank pays Shivani 3 dscoin
Block 2 now holds Frank pays Shivani 3 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
2
Verifying entire chain
..Improper hash on node 2 Does not begin with 000
Chain verification: false
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

```
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
5
Repairing the entire chain
Total execution time required to repair the chain was 642 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
View the Blockchain
```





2. Task 0 Block.java

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 24th October 2021
 *
 * Explanation and code taken from JavaDoc -
 * https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/Block.html#getNonce()
 * 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.
 * */

import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;

public class Block {
    private int index;
    private Timestamp timestamp;
    private String data;
    private int difficulty;
    private BigInteger nonce;
    private String previousHash;

    /* constructor:
    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.
    nonce - 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.
     */
    Block (int index, Timestamp timestamp, String data, int difficulty){
        this.index=index;
        this.timestamp = timestamp;
        this.data= data;
        this.difficulty=difficulty;
        this.nonce = new BigInteger("0");
    }

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

    // This method computes a hash of the concatenation of the index,
    // timestamp, data, previousHash, nonce, and difficulty.
    public String calculateHashes(){
        String block =
index+String.valueOf(timestamp)+data+previousHash+nonce+difficulty;
        byte[] bytesOfBlock = block.getBytes(StandardCharsets.UTF_8);
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hashedOutput = md.digest(bytesOfBlock);
            return bytesToHex(hashedOutput);
```

```
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return block;
    }

    // Code from https://stackoverflow.com/questions/9655181/how-to-convert-
a-byte-array-to-a-hex-string-in-java
    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);
    }

    // getter for nonce
    public BigInteger getNonce() {
        return this.nonce;
    }

    // setter for nonce
    public void setNonce(BigInteger nonce) {
        this.nonce=nonce;
    }

    // The proof of work methods finds a good hash, i.e, the number of zeroes
in the
    // start of hash must be equal to difficulty of the block
    public String proofOfWork() {
        String hashReturned = "";
        boolean proofOfWorkPass = false;
        while(!proofOfWorkPass) {
            hashReturned = calculateHashes();
            char[] charsHash = hashReturned.toCharArray();
            int difficulty = getDifficulty();
            int numberOfZero = 0;
            // count zeroes
            for (int i = 0; i < difficulty; i++) {
                if (charsHash[i] == '0') {
                    numberOfZero++;
                }
            }
            if (numberOfZero == difficulty) proofOfWorkPass = true;
            else setNonce(getNonce().add(BigInteger.valueOf(1)));
        }
        return hashReturned;
    }

    // getter for difficulty
    public int getDifficulty() {
        return this.difficulty;
    }
```

```
    // setter for difficulty
    public void setDifficulty (int difficulty) {
        this.difficulty = difficulty;
    }

    // toString method of individual block to form string in json format
    public String toString() {
        String blockString = String.format("{\"index\" : %d,\"time stamp \" :
\"%s\",\"Tx \" : \"%s\",\"PrevHash\" : \"%s\",\"nonce\" : %d,\"difficulty\" :
%d}",
                index, timestamp, data, previousHash, nonce, difficulty);
        return blockString;
    }

    // setter for previous hash
    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }
    // getter for previous hash
    public String getPreviousHash() {
        return this.previousHash;
    }
    // getter for index
    public int getIndex() {
        return this.index;
    }

    // setter for index
    public void setIndex (int index) {
        this.index =index;
    }

    // setter for timestamp
    public void setTimestamp (Timestamp timestamp) {
        this.timestamp= timestamp;
    }

    // getter for timestamp
    public Timestamp getTimestamp() {
        return this.timestamp;
    }

    // getter for data/transaction
    public String getData() {
        return this.data;
    }

    // setter for data/transaction
    public void setData (String data) {
        this.data = data;
    }
}
```

3. Task 0 BlockChain.java

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 24th October 2021
 *
 * Explanation and code taken from JavaDoc -
 * https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html#isChainValid()
 * This is the code for 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 previous hash and a difficulty of 2. On start up, this code 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 three,
 * then all hashes will begin with 3 or more zero hex digits (or 3 nibbles,
or 12 zero bits).
 * It is menu-driven and will continuously provide the user with seven
options:View basic blockchain
 * status, add a transaction to the blockchain, verify the blockchain, view
the blockchain, corrupt the
 * chain, hide the corruption by repairing the chain, exit. Based on the user
selection, certain
 * operations are performed on the blockchain.
 * */

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.Scanner;

public class BlockChain {
    // ArrayList to hold Blocks
    private final ArrayList<Block> blocks;
    // holds a SHA256 hash of the most recently added Block.
    private String chainHash;
    // Store hashes per second
    private static int hashesPerSecond;
    // store node having wrong hash
    private int incorrectNode;

    // constructor
    BlockChain() {
        // initialize blocks ArrayList, chainHash and hashesPerSecond
        blocks= new ArrayList<>();
        chainHash = "";
        hashesPerSecond=0;
    }
```

```
    /*
    * On startup the genesis block is created and added to the blockchain.
The user is then
    * prompted to select an option from the menu. Based on the user
selection, operations are
    * performed as results are displayed. Continue the process until the user
selects 6,i.e,Exit
    *
    * Based on the experiments conducted the computation time increases as
the difficulty associated
    * with the block increases. For instance, for difficulty 2, addBlock()
methods takes between
    * 0-15 milliseconds approximately. For difficulty 5, addBlock() methods
took 461 milliseconds
    * approximately. For difficulty 6, addBlock() methods took 6000
milliseconds approximately.
    * The repairChain() and isChainValid methods take approximately 0
milliseconds if all the blocks
    * have correct data irrespective of the difficulty of the blocks.
However, in the presence of a
    * malicious block having corrupt data, repairChain() method takes longer
time if there are blocks
    * with higher difficulty. For instance, if malicious block has a
difficulty of 2, the
    * repairChain() method took 2775 milliseconds. But if malicious block has
a difficulty of 6, the
    * repairChain() method took 13221 milliseconds. isChainValid() method
also takes longer time
    * if malicious block has higher difficulty. For difficulty 2 of malicious
block, isChainValid()
    * method took close to 0 milliseconds and for difficulty 6, method took
close to 362 milliseconds.
    * */
    public static void main (String[] args) {
        BlockChain blockChain = new BlockChain();
        Block genesisBlock = new Block(0, new Timestamp(new
Date().getTime()), "Genesis", 2);
        genesisBlock.setPreviousHash("");
        blockChain.computeHashesPerSecond();
        blockChain.addBlock(genesisBlock);
        int userInput = -1;
        while(userInput!=6) {
            displayMenu();
            Scanner getInput = new Scanner(System.in);
            userInput = Integer.parseInt(getInput.nextLine());
            switch (userInput) {
                // display basic blockchain status
                case 0 -> {
                    System.out.println("Current size of chain: " +
blockChain.getChainSize());
                    System.out.println("Difficulty of most recent block: " +
blockChain.getLatestBlock().getDifficulty());
                    System.out.println("Total difficulty for all blocks: " +
blockChain.getTotalDifficulty());
                    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.chainHash);
                }
                // fetch difficulty, transaction data from user, use this
data to create a new block
                // and add new block created to the blockchain. Calculate the
time taken for this
                // operation
                case 1 -> {
                    System.out.println("Enter difficulty > 0");
                    int difficulty = Integer.parseInt(getInput.nextLine());
                    System.out.println("Enter transaction");
                    String transaction = getInput.nextLine();
                    long startTime = System.currentTimeMillis();
                    blockChain.addBlock(new
Block(blockChain.getLatestBlock().getIndex() + 1,
                            new Timestamp(new Date().getTime()), transaction,
difficulty));
                    long endTime = System.currentTimeMillis();
                    System.out.println("Total execution time to add this
block was " + (endTime - startTime) + " milliseconds");
                }
                // check if the chain is correct or not and display to user.
                // If chain is incorrect, display the incorrect node
information
                // to the user along with the total execution time for the
checking process
                case 2 -> {
                    System.out.println("Verifying entire chain");
                    long startTime = System.currentTimeMillis();
                    boolean chainVerified = blockChain.isChainValid();
                    long endTime = System.currentTimeMillis();
                    if (!chainVerified) {
                        int difficulty =
blockChain.getBlock(blockChain.incorrectNode).getDifficulty();
                        System.out.printf("..Improper hash on node %d Does
not begin with %s\n", blockChain.incorrectNode, "0".repeat(Math.max(0,
difficulty)));
                    }
                    System.out.println("Chain verification: " +
chainVerified);
                    System.out.println("Total execution time required to
verify the chain was " + (endTime - startTime) + " milliseconds");
                }
                // display the blockchain to the user in json format
                case 3 -> System.out.println(blockChain);
                // fetch id, and transaction from the user and corrupt the
blockchain with
                // this data
                case 4 -> {
                    System.out.println("corrupt the Blockchain\n" +
                            "Enter block ID of block to corrupt");
                    int blockId = Integer.parseInt(getInput.nextLine());
```

```
                    System.out.println("Enter new data for block " +
blockId);
                    String blockTransaction = getInput.nextLine();
                    blockChain.getBlock(blockId).setData(blockTransaction);
                    System.out.printf("Block %d now holds %s\n", blockId,
blockTransaction);
                }
                // repair the blockchain and display the total execution time
for the repairing process
                case 5 -> {
                    System.out.println("Repairing the entire chain");
                    long startTime = System.currentTimeMillis();
                    blockChain.repairChain();
                    long endTime = System.currentTimeMillis();
                    System.out.printf("Total execution time required to
repair the chain was %d milliseconds\n", endTime - startTime);
                }
            }
        }
    }

    // check the proofOfWork for each block and add it to the blockchain,
update
    // chain hash with the latest added block's proofOfWork
    public void addBlock (Block newBlock) {
        if(blocks.size()!=0) newBlock.setPreviousHash(getBlock(blocks.size()-
1).proofOfWork());
        blocks.add(newBlock);
        chainHash = newBlock.proofOfWork();
    }

    // getter for timestamp
    public Timestamp getTime() {
        return new Timestamp(new Date().getTime());
    }

    // getter for latest block
    public Block getLatestBlock() {
        return getBlock(blocks.size()-1);
    }

    // getter for chain size
    public int getChainSize() {
        return blocks.size();
    }

    /*
    * This method computes exactly 1 million hashes and times how long that
process takes.
    * So, hashes per second is approximated as (1 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() {
        String sampleString = "00000000";
        long startTime = getTime().getTime();
```

```
        byte[] bytesOfHash = sampleString.getBytes(StandardCharsets.UTF_8);
        for(int i=0; i<1000000;i++) {
            try {
                // Use SHA-256 for hashing
                MessageDigest md = MessageDigest.getInstance("SHA-256");
                md.digest(bytesOfHash);
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
        }
        long endTime = getTime().getTime();
        BlockChain.hashesPerSecond = (int) (1000000*1000/ (double) (endTime -
startTime));
    }

    public int getHashesPerSecond() {
        return hashesPerSecond;
    }

    // creates string in json format for entire blockchain using toString
method of
    // individual blocks
    public String toString() {
        System.out.println("View the Blockchain");
        StringBuilder jsonString= new StringBuilder("{\"ds_chain\":[");
        for(int i=0; i<blocks.size();i++) {
            Block block = getBlock(i);
            String blockString;
            if(i!= blocks.size()-1) {
                blockString = block+",\n";
            } else {
                blockString =
String.format("%s],\n\"chainHash\":\"%s\"}",block,chainHash);
            }
            jsonString.append(blockString);
        }
        return jsonString.toString();
    }

    // getter for block at index i
    public Block getBlock (int i) {
        return blocks.get(i);
    }

    // compute and return the total difficulty of all blocks on the chain.
    public int getTotalDifficulty() {
        int totalDifficulty=0;
        for(int i=0; i< blocks.size(); i++) {
            totalDifficulty+=getBlock(i).getDifficulty();
        }
        return totalDifficulty;
    }

    // Compute and return the expected number of hashes required for the
entire chain.
    public double getTotalExpectedHashes() {
        double totalExpectedHashes = 0;
```

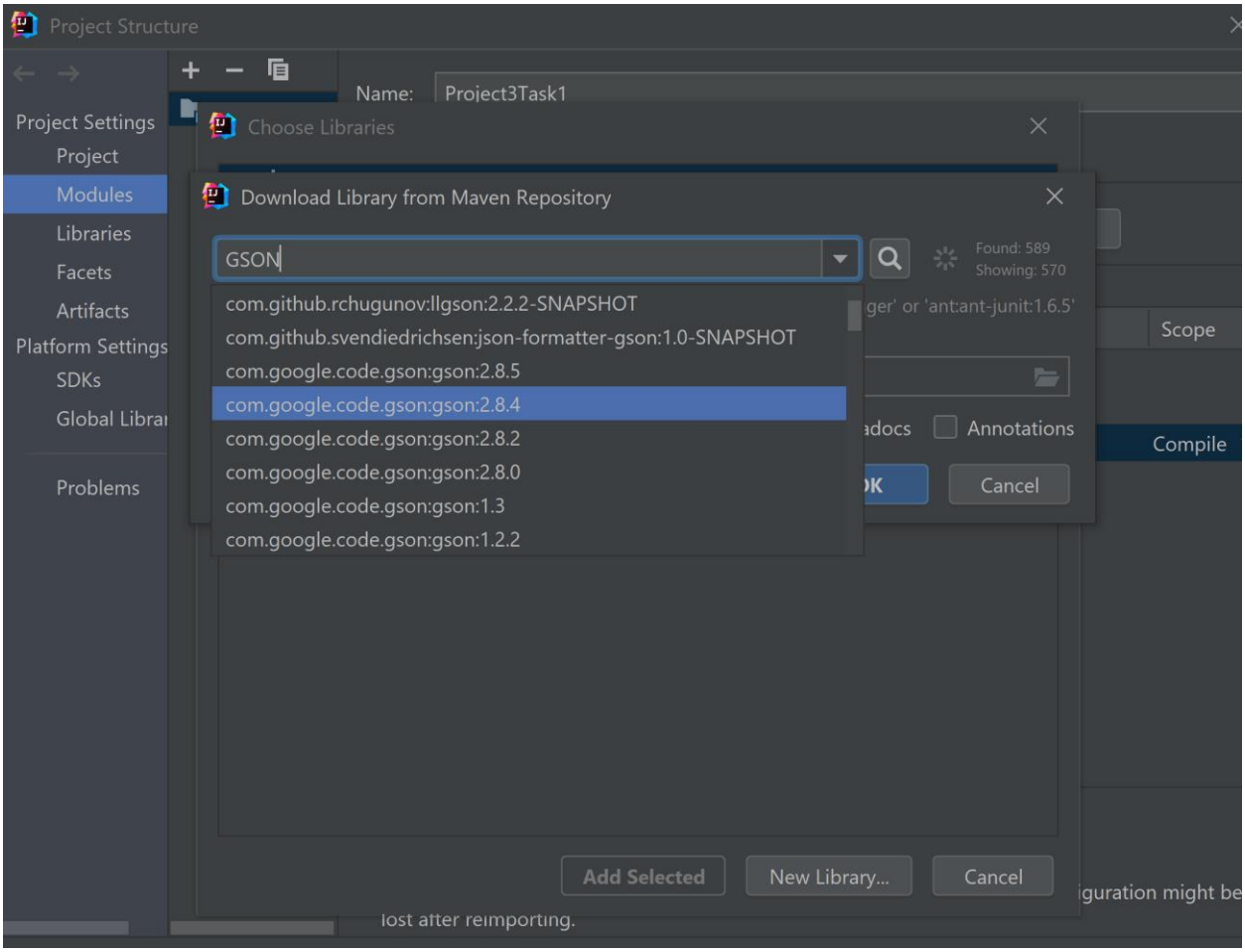
```
        for(int i=0; i<blocks.size();i++) {
            totalExpectedHashes +=
Math.pow((16),getBlock(i).getDifficulty());
        }
        return totalExpectedHashes;
    }

    /*
    * If the chain only contains one block, the genesis block at position 0,
this method 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 false.
Otherwise, return true.
    * If the chain has more blocks than one, begin checking all blocks using
chainValidation
    * method. If chainValidation method returns a non-negative number
continue checking the
    * blocks in the blockchain. When chainValidation method returns -1, then
it means the
    * chain validation fails. */
    public boolean isChainValid() {
        if(blocks.size()==1) {
            Block genesis = getBlock(0);
            String genesisHash = genesis.proofOfWork();
            int numberOfZero = 0;
            int difficulty = genesis.getDifficulty();
            char[] charsHash = genesisHash.toCharArray();
            for (int i = 0; i < difficulty; i++) {
                if (charsHash[i] == '0') numberOfZero++;
            }
            return chainHash.equals(genesisHash) && numberOfZero ==
difficulty;
        } else {
            for(int i=0; i<blocks.size();i++) {
                // checks each node
                int node = chainValidation(i);
                if(node==-1) {
                    // setting index of incorrect node with wrong hash value
                    incorrectNode=i-1;
                    return false;
                }
            }
            return chainHash.equals(getLatestBlock().proofOfWork());
        }
    }

    /*
     * Check hash pointer of each Block and compare with proofOfWork of
previous block.
     * If they match and if the proof of work is correct, return block index.
else
     * return -1; */
    public int chainValidation(int blockIndex) {
        String prevBlockHash = blockIndex==0?
```

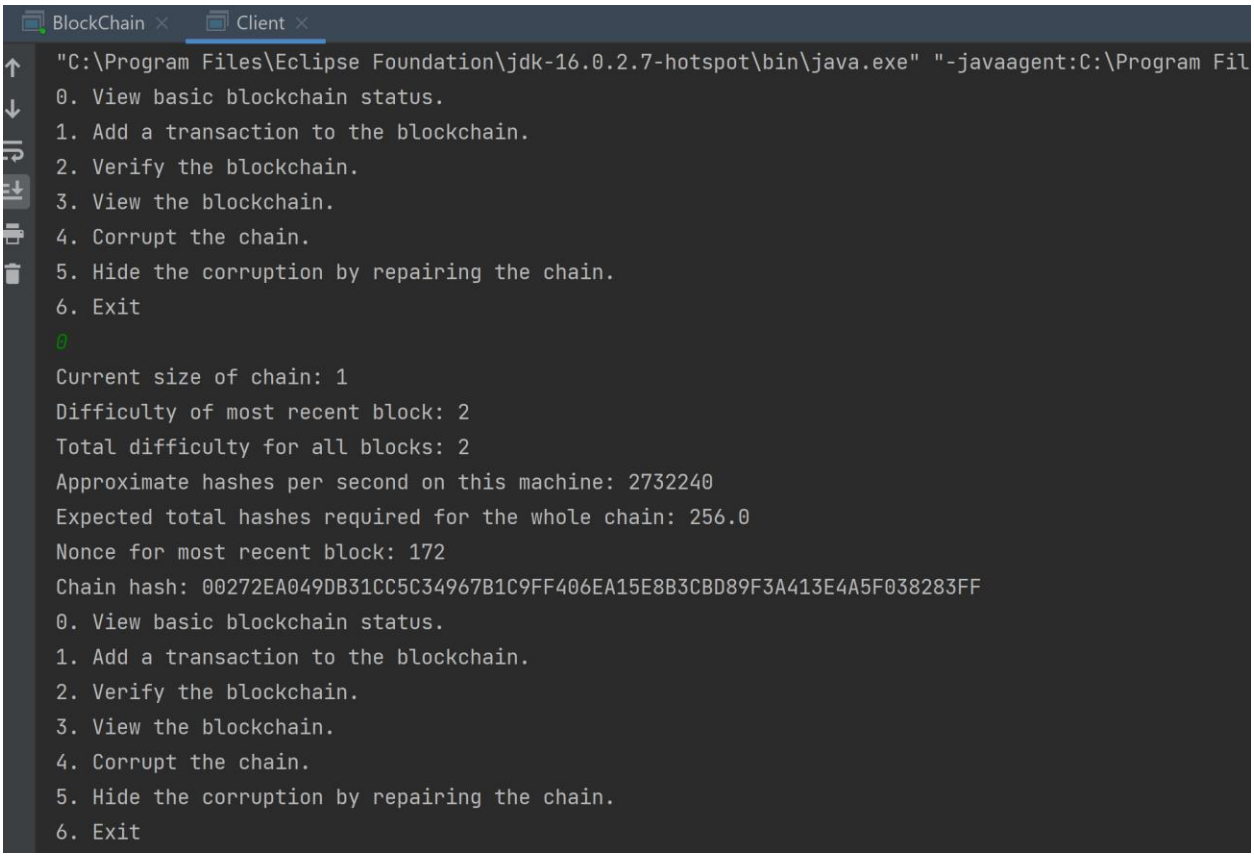
```
                "":getBlock(blockIndex-1).proofOfWork();
        String hashPointer = getBlock(blockIndex).getPreviousHash();
        if(prevBlockHash.equals(hashPointer)) {
            if(blockIndex!=0){
                char[] charsHash = prevBlockHash.toCharArray();
                int difficulty = getBlock(blockIndex-1).getDifficulty();
                int numberOfZero = 0;
                for (int i = 0; i < difficulty; i++) {
                    if (charsHash[i] == '0') numberOfZero++;
                }
                if (numberOfZero == difficulty) return blockIndex;
            }
            return blockIndex;
        }
        return -1;
    }

    // checks each block and recomputes proofOfWork for each block and
    // assigns the correct hash pointer in the next block as well as the
    // chain hash using the proofOfWork of the last added block.
    public void repairChain() {
        for(int i=0; i<getChainSize();i++) {
            if(i==0) getBlock(i).setPreviousHash("");
            String correctHash = getBlock(i).proofOfWork();
            if(i+1 <
getChainSize())getBlock(i+1).setPreviousHash(correctHash);
            if(i==getChainSize()-1) chainHash=correctHash;
        }
        chainHash=getLatestBlock().proofOfWork();
    }

    // menu options to be displayed to user
    private static void displayMenu() {
        System.out.println("""
                0. View basic blockchain status.
                1. Add a transaction to the blockchain.
                2. Verify the blockchain.
                3. View the blockchain.
                4. Corrupt the chain.
                5. Hide the corruption by repairing the chain.
                6. Exit""");
    }

}
```

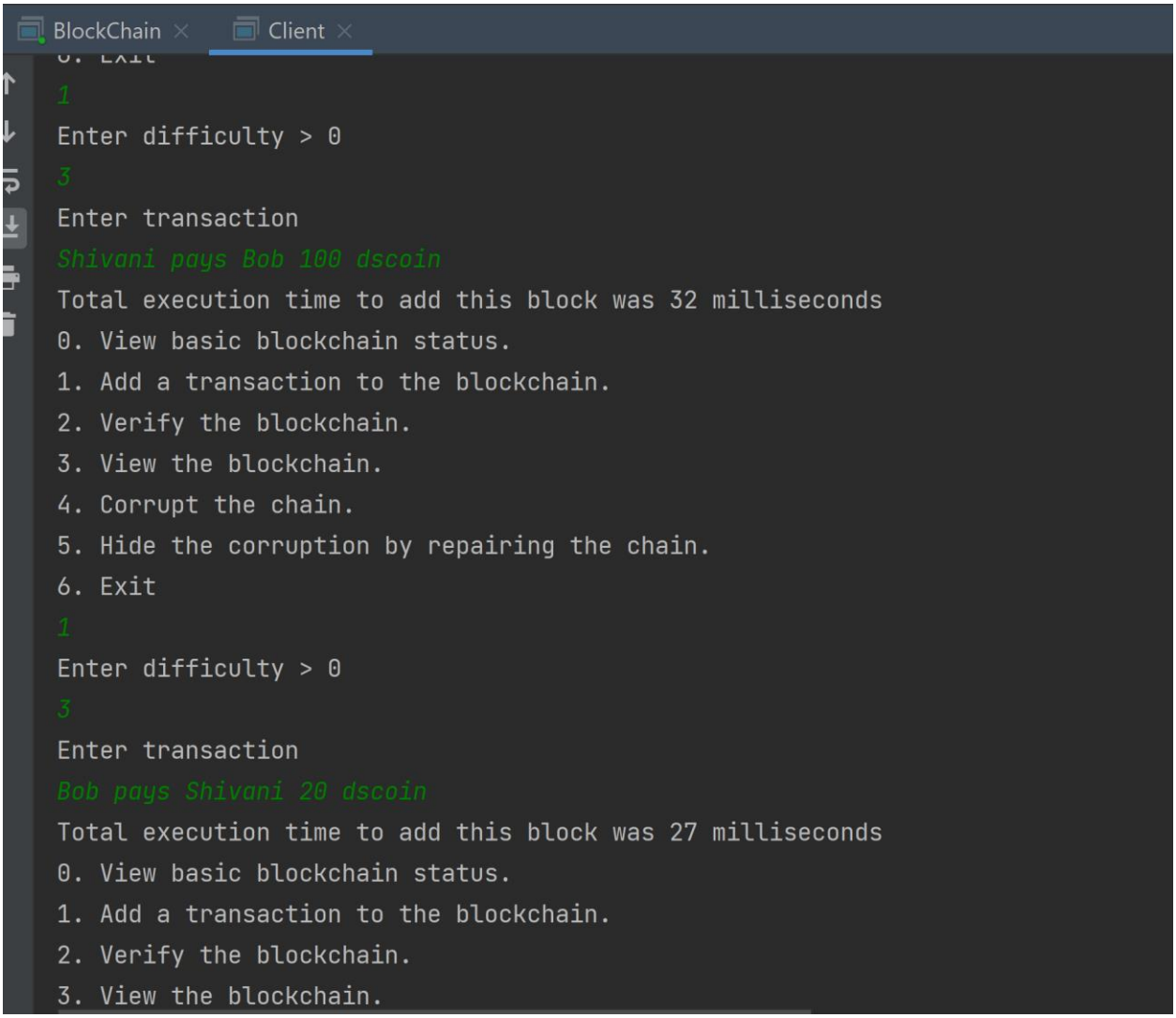
4. Task 1 Execution

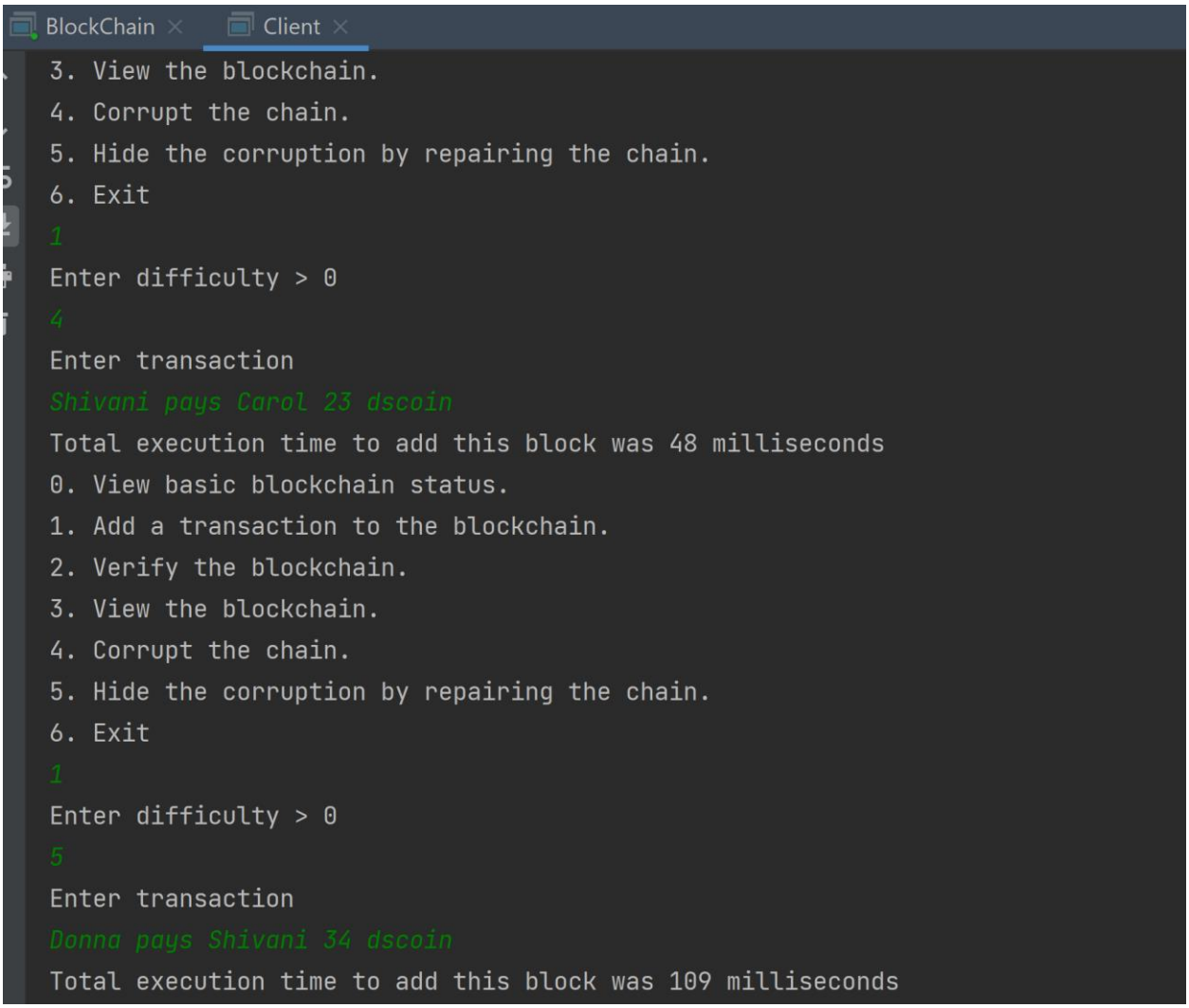
Important:

In order to parse JSON objects, GSON (version: gson-2.8.4.jar) library is added to the Maven project. Additionally, json-simple-1.1.1 jar is also added to the project to create JSON object. This jar file is added in the zip file. Please add this dependency to the Maven project as well before running the code in Task 1.


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Librar
Problems
Name:
Project3Task1
Choose Libraries
Download Library from Maven Repository
GSON
Found: 589
Showing: 570
com.github.rchugunov:llgson:2.2.2-SNAPSHOT
com.github.svendiedrichsen:json-formatter-gson:1.0-SNAPSHOT
com.google.code.gson:gson:2.8.5
com.google.code.gson:gson:2.8.4
com.google.code.gson:gson:2.8.2
com.google.code.gson:gson:2.8.0
com.google.code.gson:gson:1.3
com.google.code.gson:gson:1.2.2
ger' or 'ant:ant-junit:1.6.5'
Scope
Annotations
Compile
Cancel
Add Selected
New Library...
Cancel
guration might be
lost after reimporting.

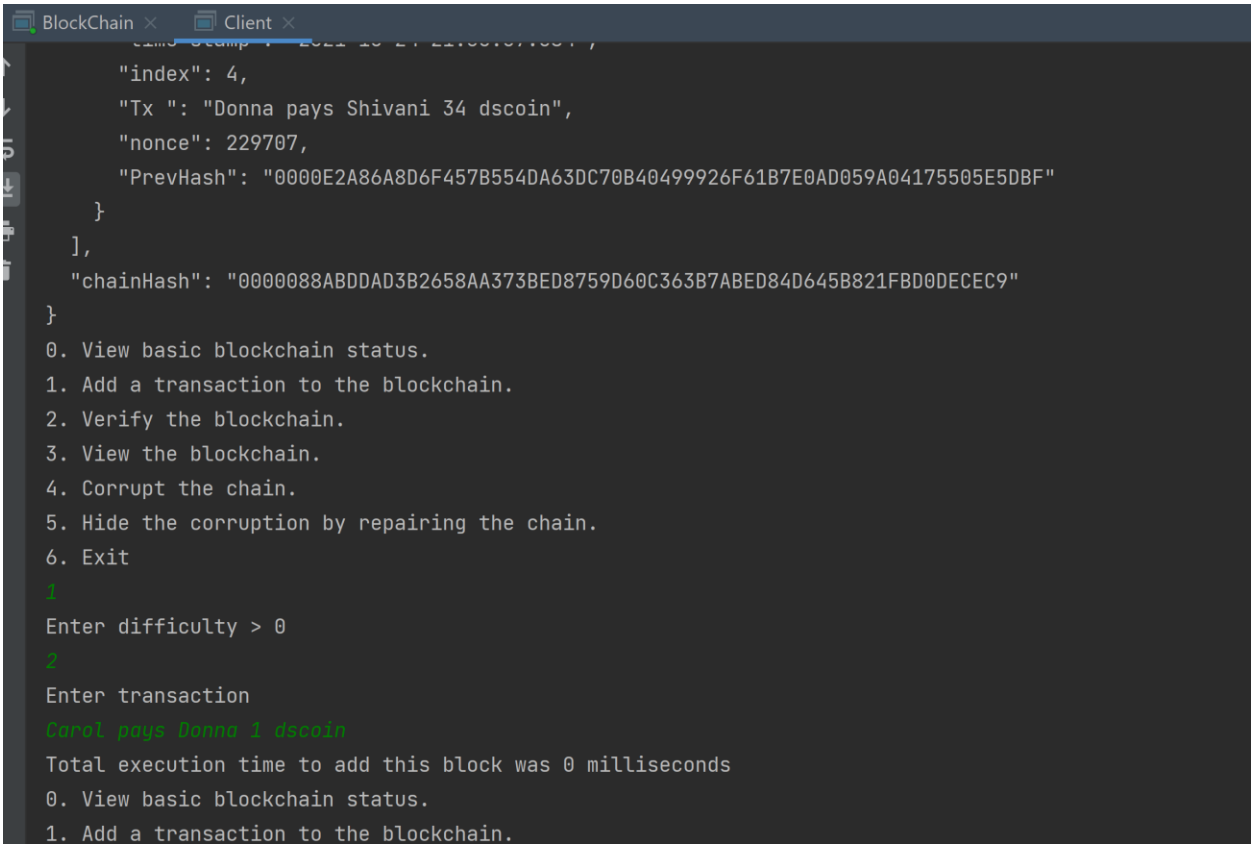

BlockChain
Client
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.exe" "-javaagent:C:\Program Fil
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
0
Current size of chain: 1
Difficulty of most recent block: 2
Total difficulty for all blocks: 2
Approximate hashes per second on this machine: 2732240
Expected total hashes required for the whole chain: 256.0
Nonce for most recent block: 172
Chain hash: 00272EA049DB31CC5C34967B1C9FF406EA15E8B3CBD89F3A413E4A5F038283FF
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

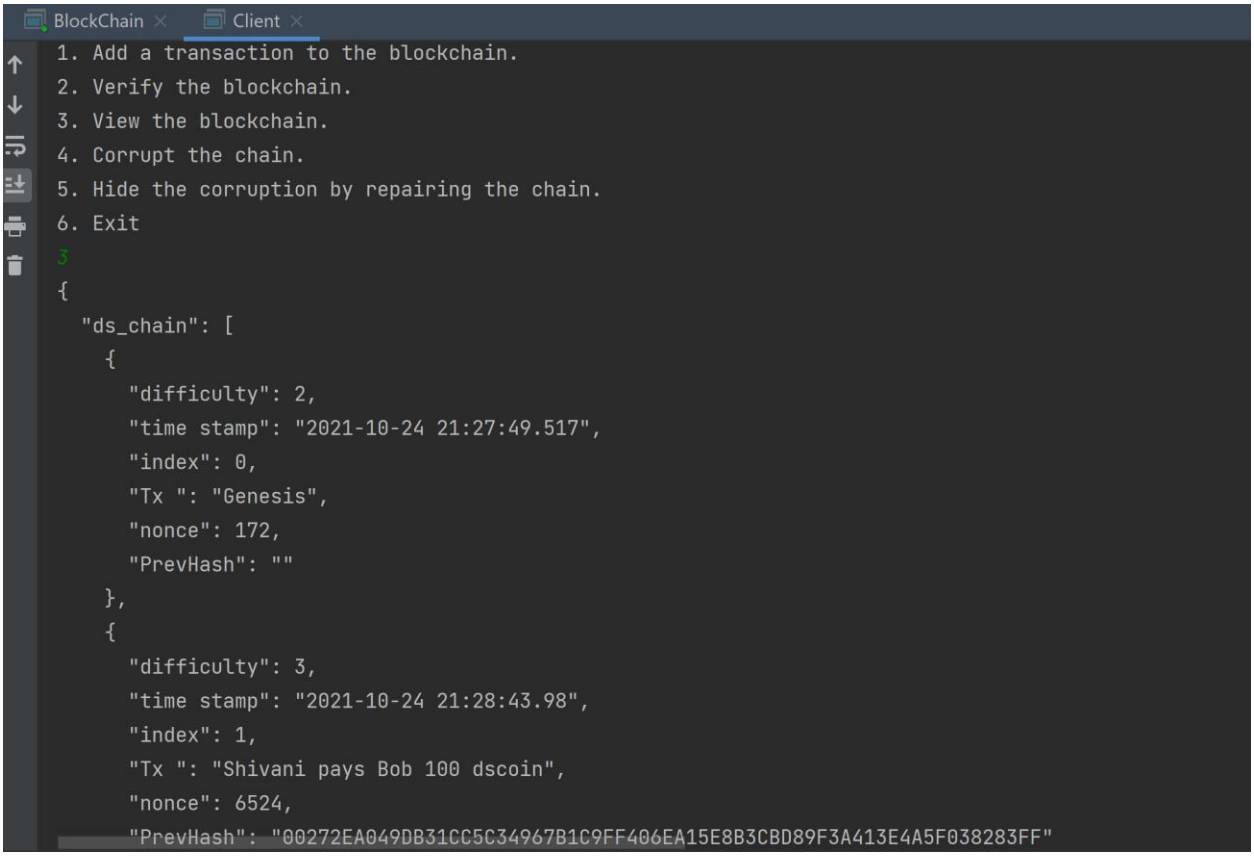

BlockChain
Client
1
Enter difficulty > 0
3
Enter transaction
Shivani pays Bob 100 dscoin
Total execution time to add this block was 32 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
3
Enter transaction
Bob pays Shivani 20 dscoin
Total execution time to add this block was 27 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.

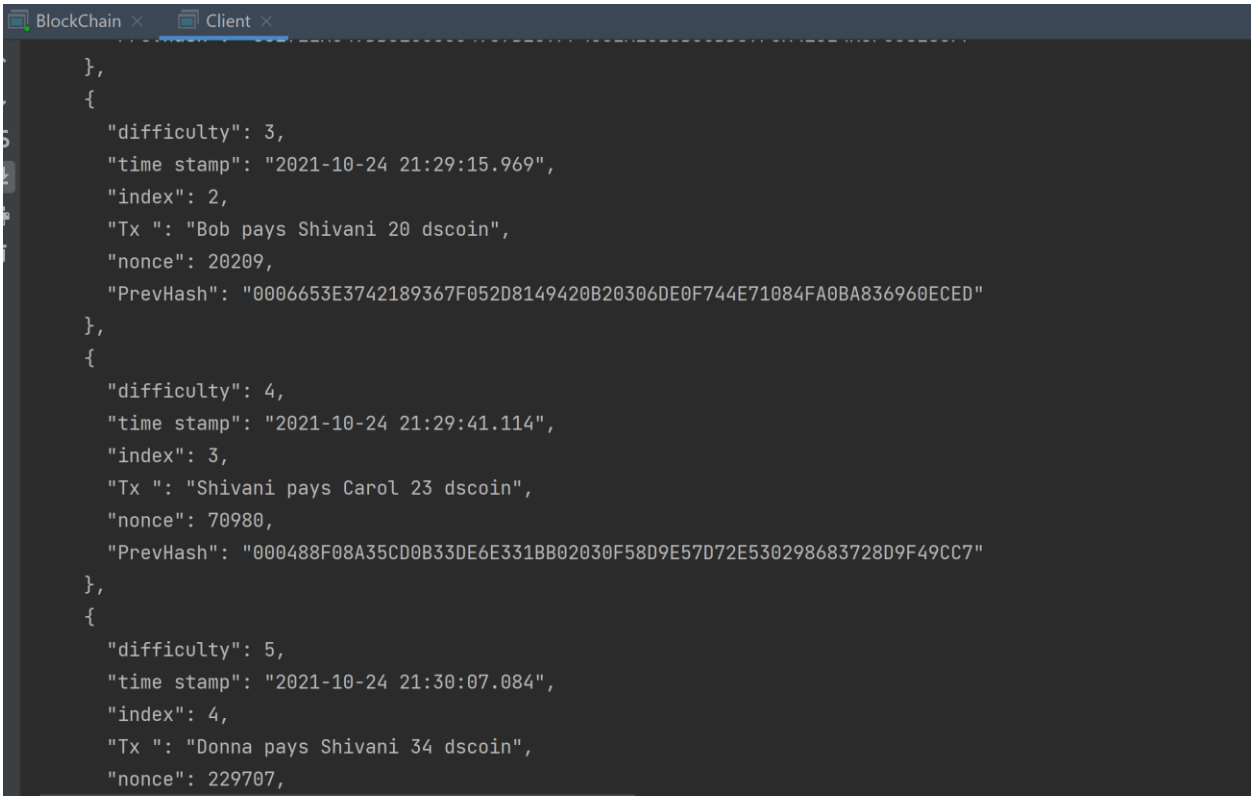

BlockChain
Client
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
4
Enter transaction
Shivani pays Carol 23 dscoin
Total execution time to add this block was 48 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
5
Enter transaction
Donna pays Shivani 34 dscoin
Total execution time to add this block was 109 milliseconds

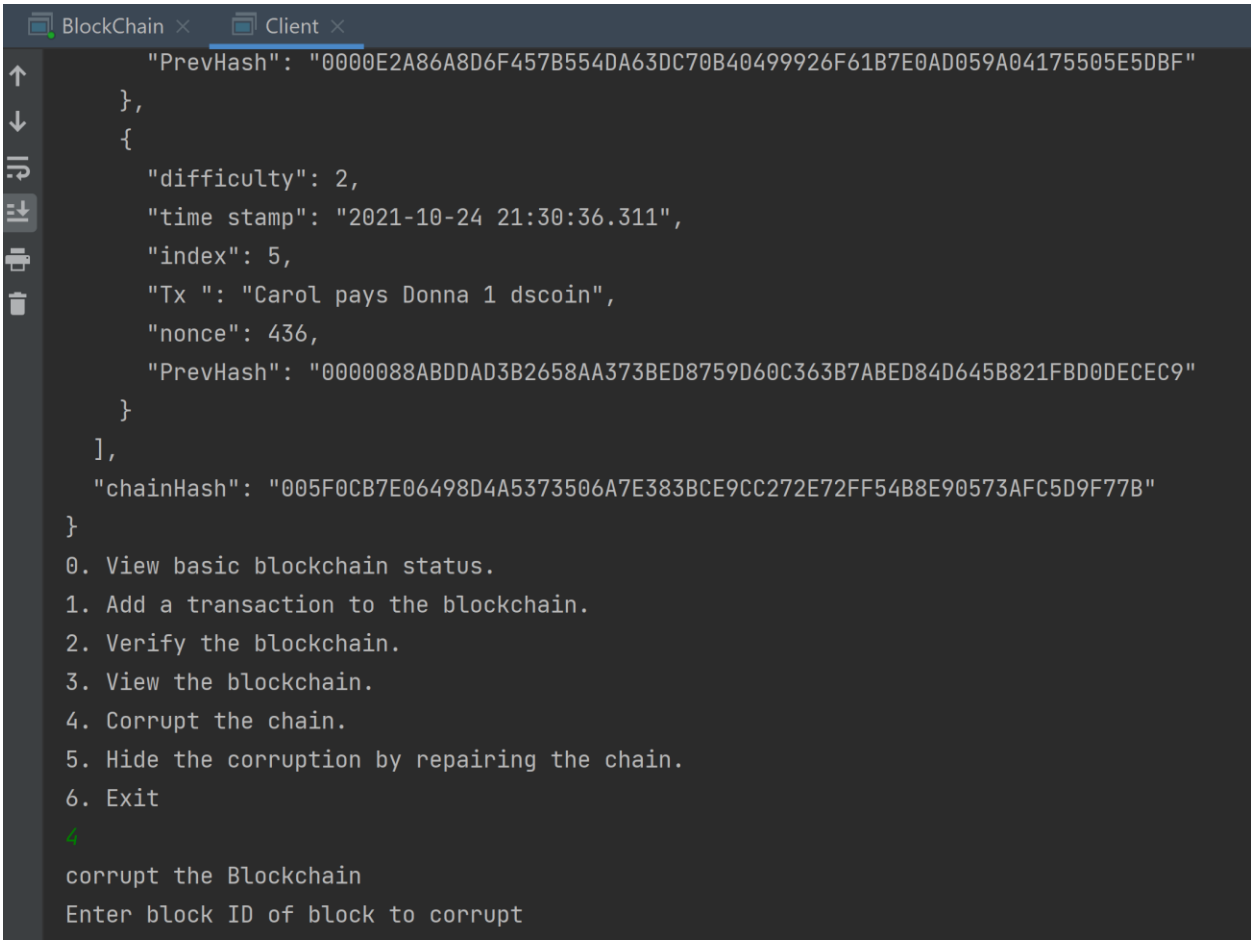
```
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
{
  "ds_chain": [
    {
      "difficulty": 2,
      "time stamp": "2021-10-24 21:27:49.517",
      "index": 0,
      "Tx ": "Genesis",
      "nonce": 172,
      "PrevHash": ""
    },
    {
      "difficulty": 3,
      "time stamp": "2021-10-24 21:28:43.98",
      "index": 1,
      "Tx ": "Shivani pays Bob 100 dscoin",
```

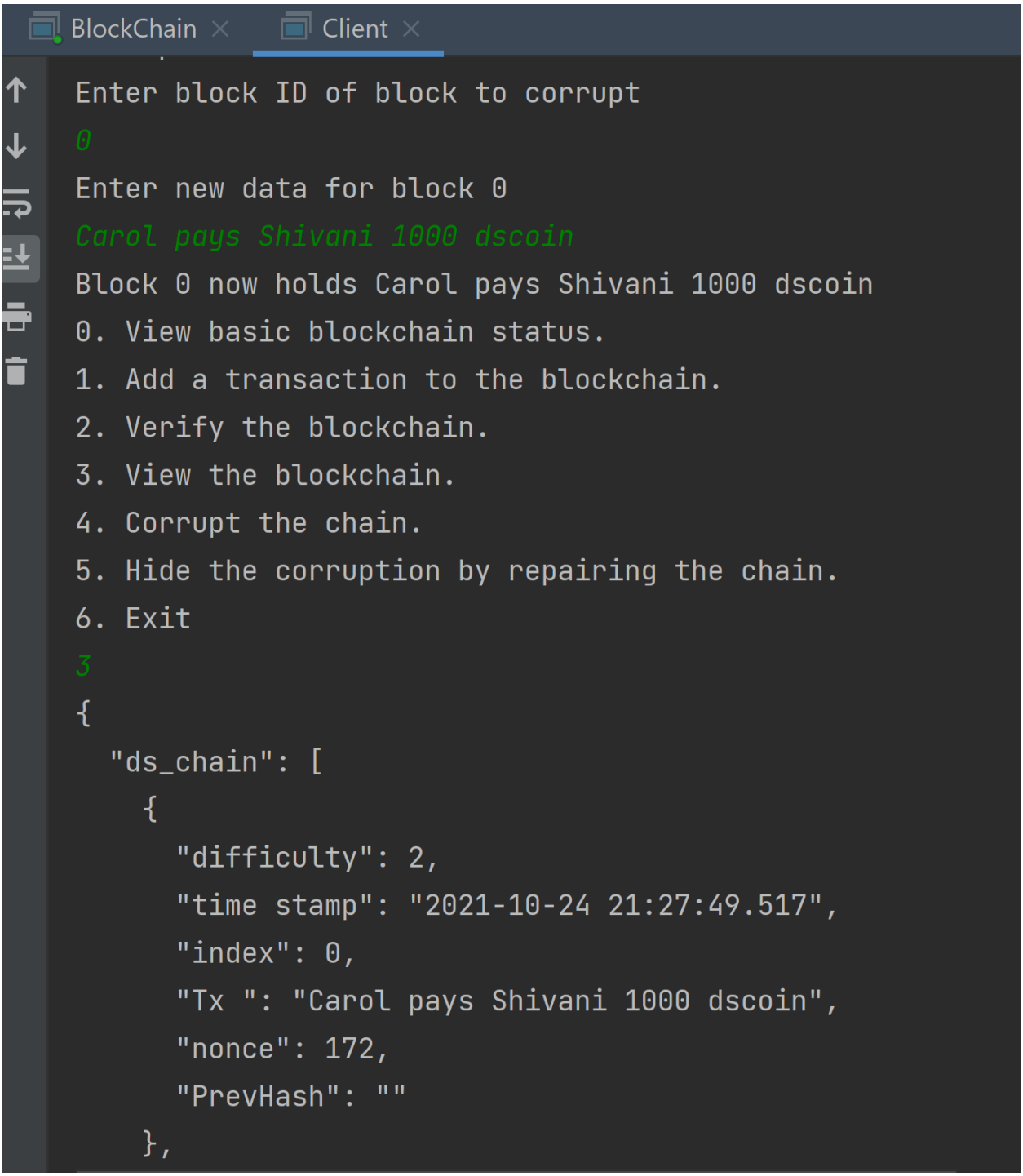
```
    "nonce": 6524,
    "PrevHash": "00272EA049DB31CC5C34967B1C9FF406EA15E8B3CBD89F3A413E4A5F038283FF"
  },
  {
    "difficulty": 3,
    "time stamp": "2021-10-24 21:29:15.969",
    "index": 2,
    "Tx ": "Bob pays Shivani 20 dscoin",
    "nonce": 20209,
    "PrevHash": "0006653E3742189367F052D8149420B20306DE0F744E71084FA0BA836960ECED"
  },
  {
    "difficulty": 4,
    "time stamp": "2021-10-24 21:29:41.114",
    "index": 3,
    "Tx ": "Shivani pays Carol 23 dscoin",
    "nonce": 70980,
    "PrevHash": "000488F08A35CD0B33DE6E331BB02030F58D9E57D72E530298683728D9F49CC7"
  },
  {
    "difficulty": 5,
    "time stamp": "2021-10-24 21:30:07.084",
```

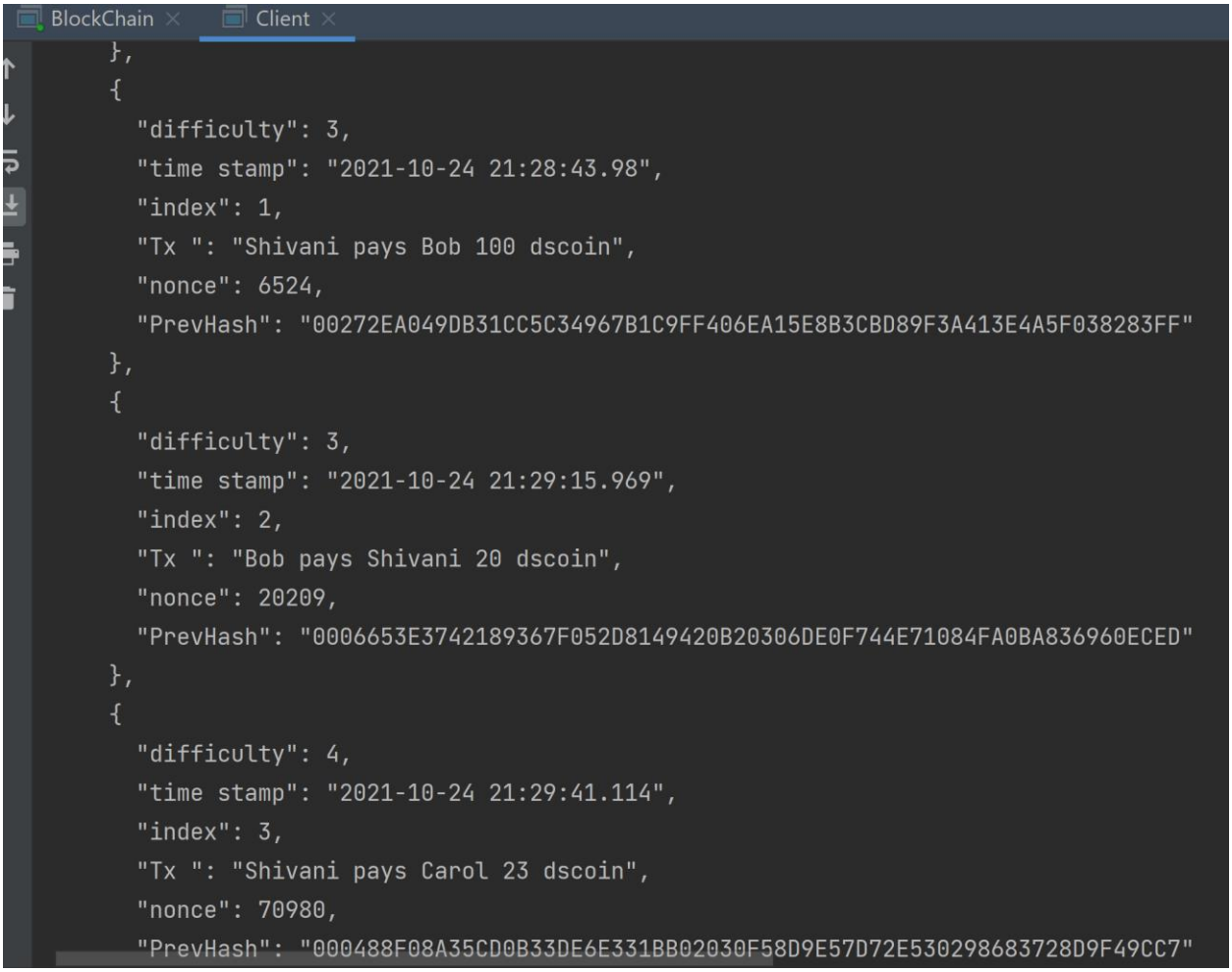



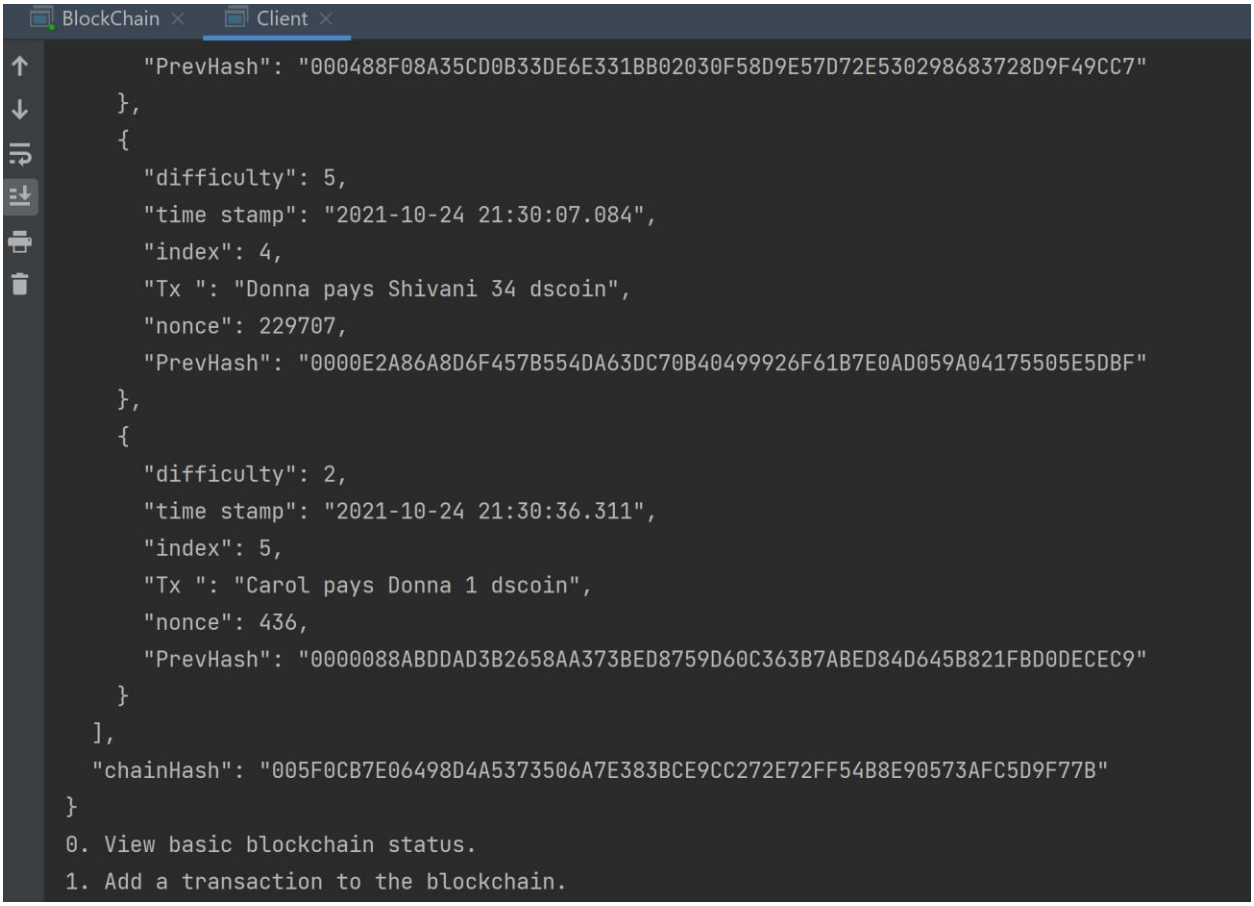

BlockChain
Client
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
{
"ds_chain": [
{
"difficulty": 2,
"time stamp": "2021-10-24 21:27:49.517",
"index": 0,
"Tx ": "Genesis",
"nonce": 172,
"PrevHash": ""
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:28:43.98",
"index": 1,
"Tx ": "Shivani pays Bob 100 dscoin",
"nonce": 6524,
"PrevHash": "00272EA049DB31CC5C34967B1C9FF406EA15E8B3CBD89F3A413E4A5F038283FF"

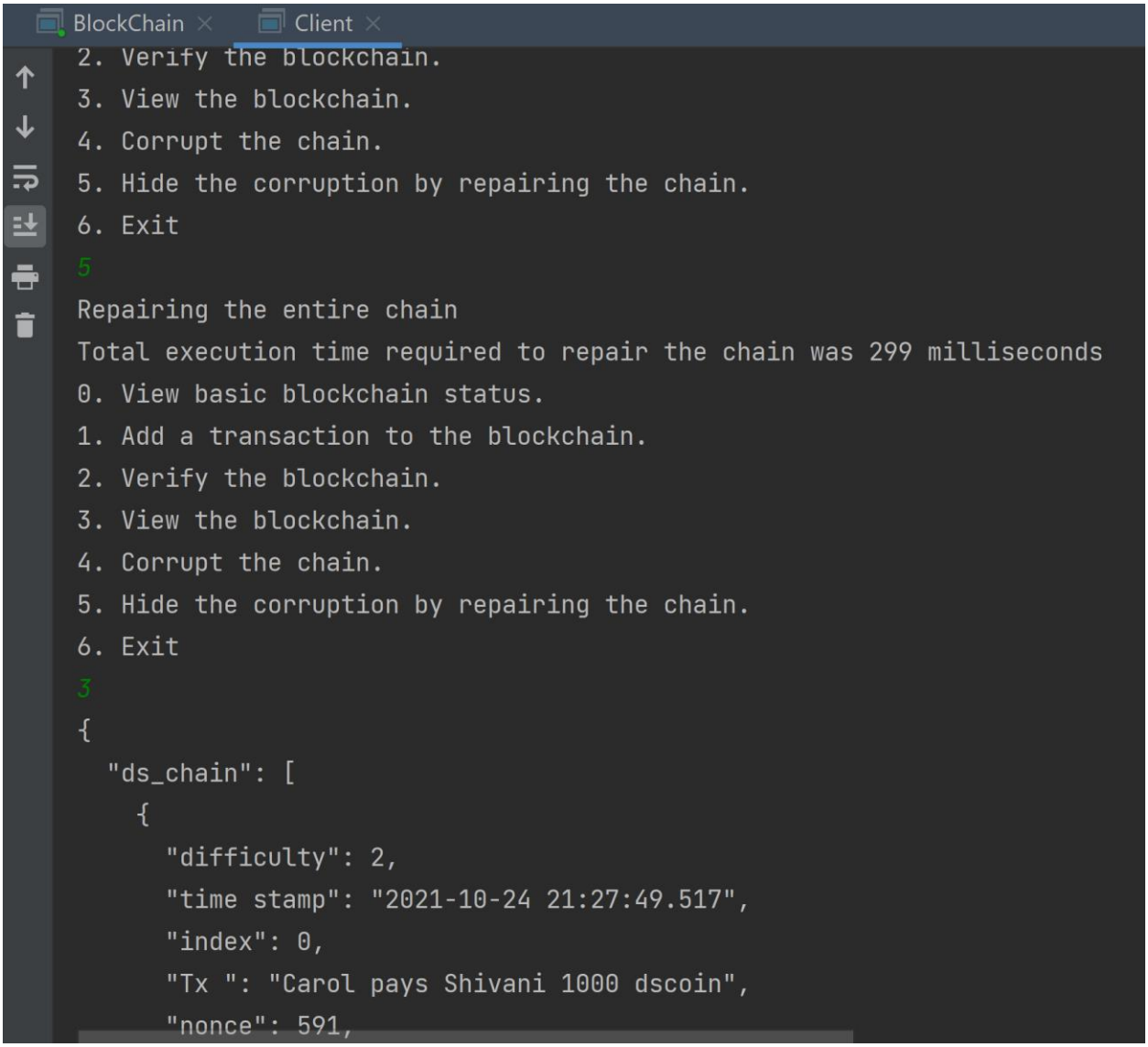

BlockChain
Client
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:29:15.969",
"index": 2,
"Tx ": "Bob pays Shivani 20 dscoin",
"nonce": 20209,
"PrevHash": "0006653E3742189367F052D8149420B20306DE0F744E71084FA0BA836960ECED"
},
{
"difficulty": 4,
"time stamp": "2021-10-24 21:29:41.114",
"index": 3,
"Tx ": "Shivani pays Carol 23 dscoin",
"nonce": 70980,
"PrevHash": "000488F08A35CD0B33DE6E331BB02030F58D9E57D72E530298683728D9F49CC7"
},
{
"difficulty": 5,
"time stamp": "2021-10-24 21:30:07.084",
"index": 4,
"Tx ": "Donna pays Shivani 34 dscoin",
"nonce": 229707,

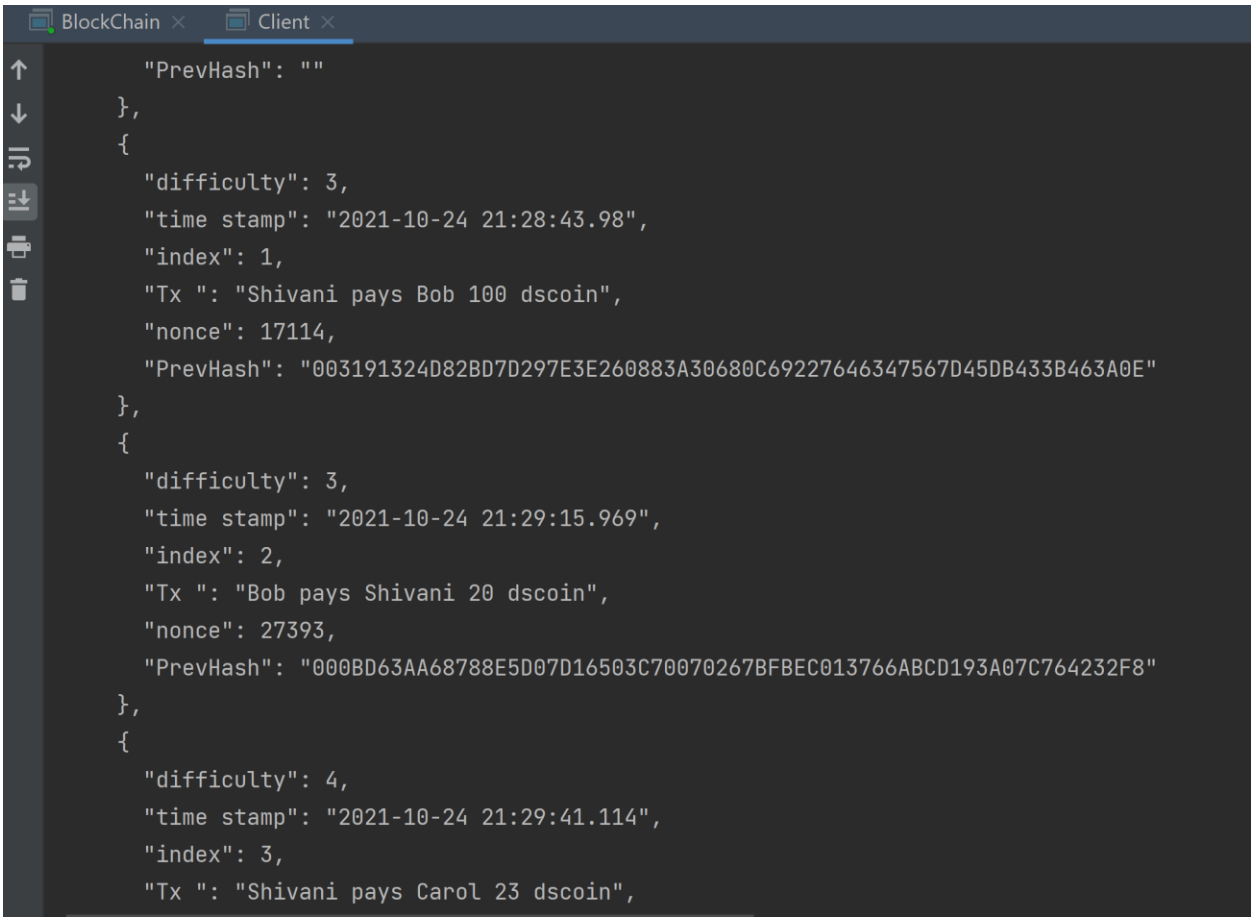

BlockChain
Client
"PrevHash": "0000E2A86A8D6F457B554DA63DC70B40499926F61B7E0AD059A04175505E5DBF"
},
{
"difficulty": 2,
"time stamp": "2021-10-24 21:30:36.311",
"index": 5,
"Tx ": "Carol pays Donna 1 dscoin",
"nonce": 436,
"PrevHash": "0000088ABDDAD3B2658AA373BED8759D60C363B7ABED84D645B821FBD0DECEC9"
}
],
"chainHash": "005F0CB7E06498D4A5373506A7E383BCE9CC272E72FF54B8E90573AFC5D9F77B"
}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
4
corrupt the Blockchain
Enter block ID of block to corrupt

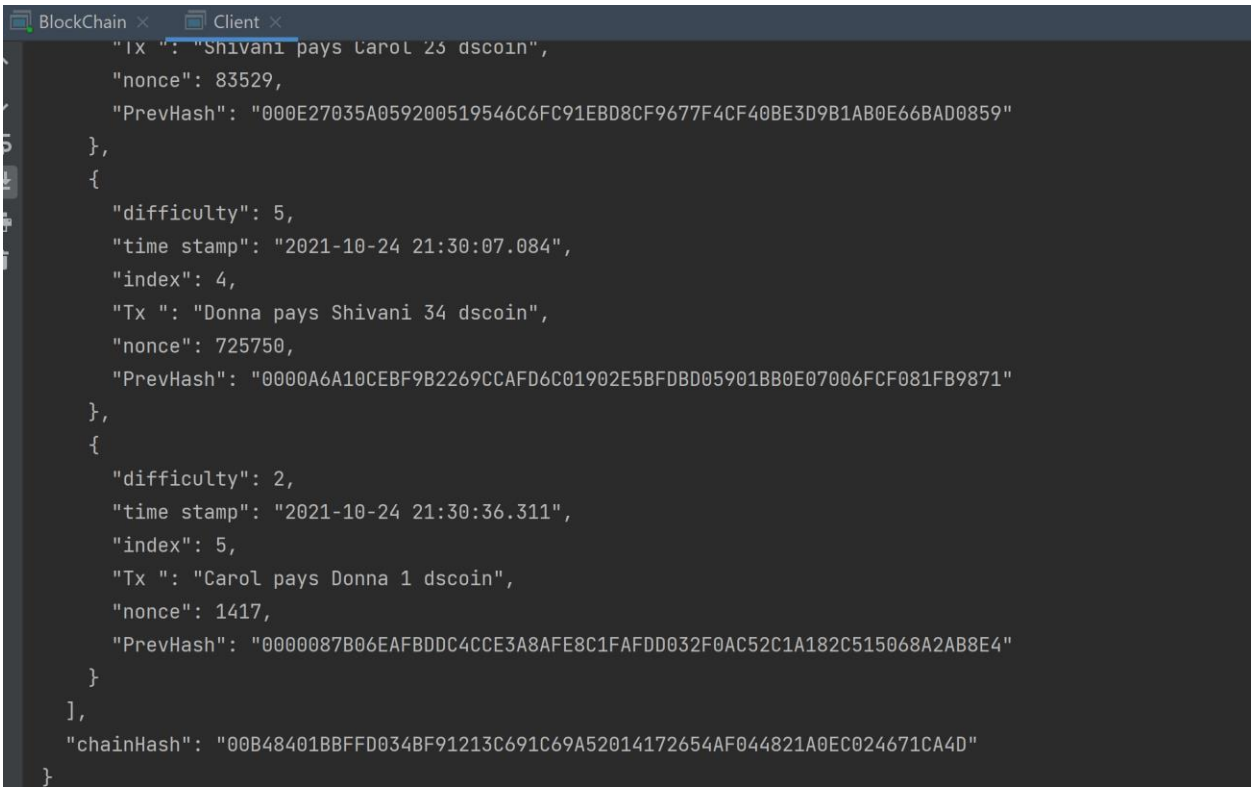

BlockChain
Client
Enter block ID of block to corrupt
0
Enter new data for block 0
Carol pays Shivani 1000 dscoin
Block 0 now holds Carol pays Shivani 1000 dscoin
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
{
"ds_chain": [
{
"difficulty": 2,
"time stamp": "2021-10-24 21:27:49.517",
"index": 0,
"Tx ": "Carol pays Shivani 1000 dscoin",
"nonce": 172,
"PrevHash": ""
},

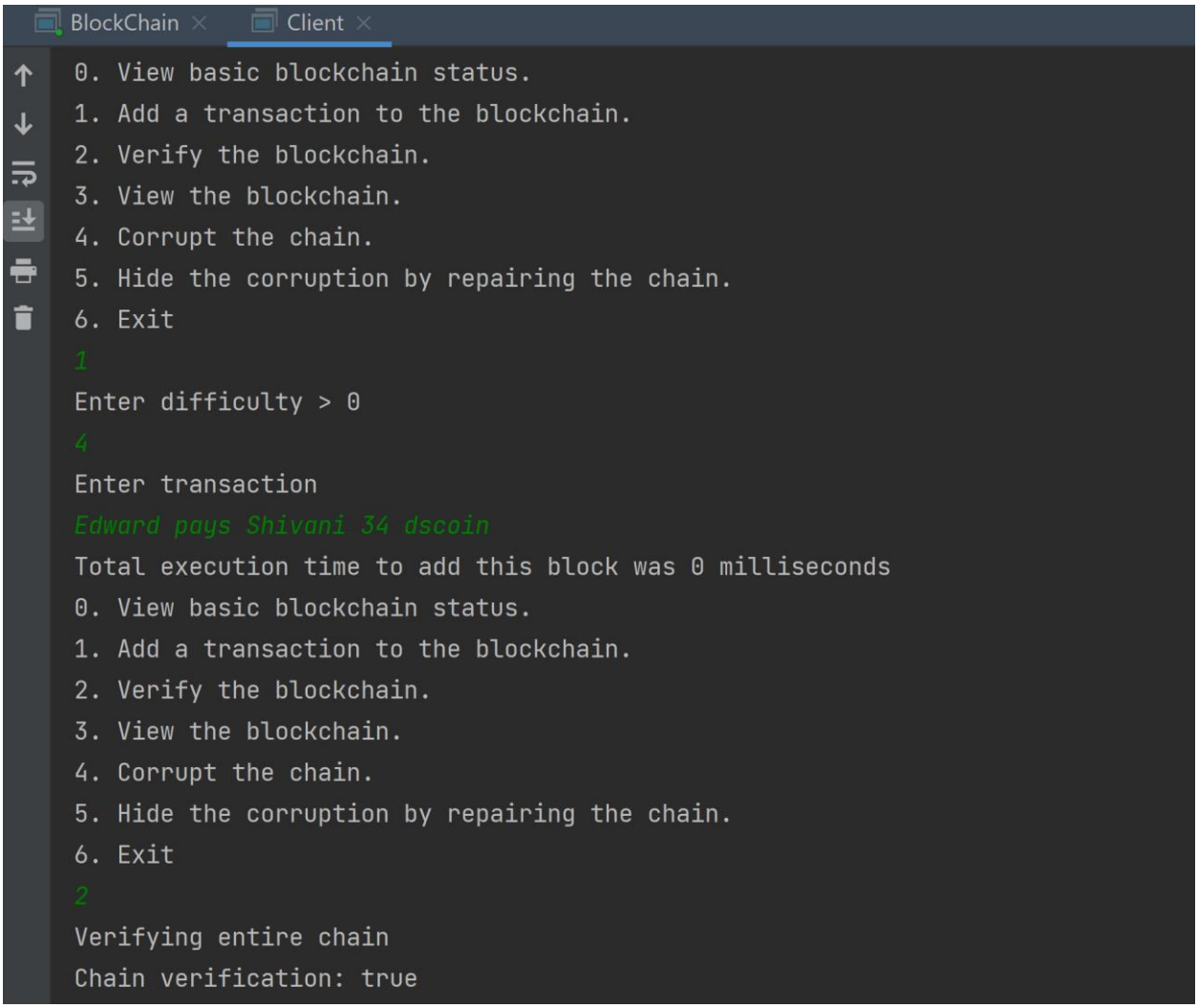

BlockChain
Client
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:28:43.98",
"index": 1,
"Tx ": "Shivani pays Bob 100 dscoin",
"nonce": 6524,
"PrevHash": "00272EA049DB31CC5C34967B1C9FF406EA15E8B3CBD89F3A413E4A5F038283FF"
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:29:15.969",
"index": 2,
"Tx ": "Bob pays Shivani 20 dscoin",
"nonce": 20209,
"PrevHash": "0006653E3742189367F052D8149420B20306DE0F744E71084FA0BA836960ECED"
},
{
"difficulty": 4,
"time stamp": "2021-10-24 21:29:41.114",
"index": 3,
"Tx ": "Shivani pays Carol 23 dscoin",
"nonce": 70980,
"PrevHash": "000488F08A35CD0B33DE6E331BB02030F58D9E57D72E530298683728D9F49CC7"

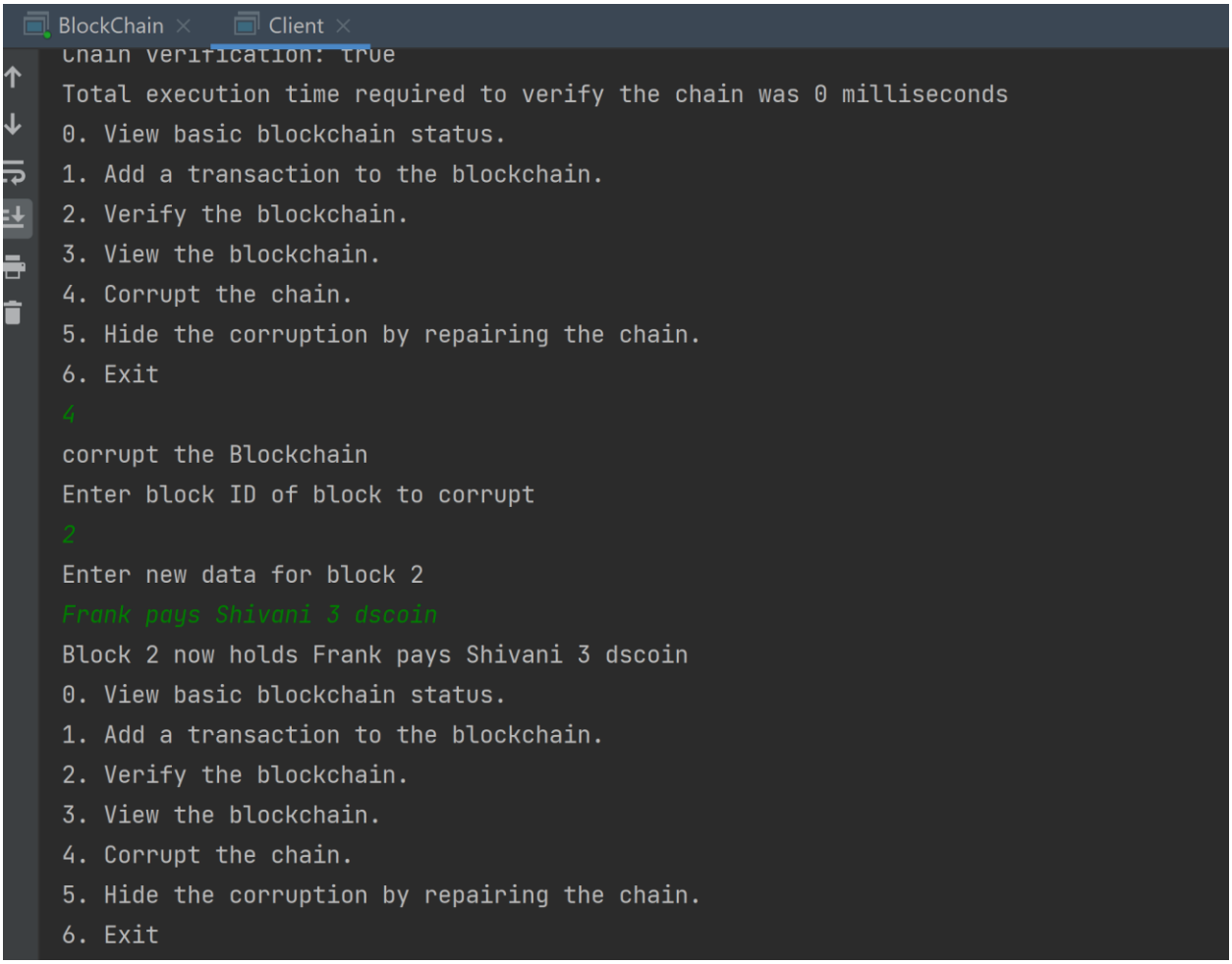

BlockChain
Client
"PrevHash": "000488F08A35CD0B33DE6E331BB02030F58D9E57D72E530298683728D9F49CC7"
},
{
"difficulty": 5,
"time stamp": "2021-10-24 21:30:07.084",
"index": 4,
"Tx ": "Donna pays Shivani 34 dscoin",
"nonce": 229707,
"PrevHash": "0000E2A86A8D6F457B554DA63DC70B40499926F61B7E0AD059A04175505E5DBF"
},
{
"difficulty": 2,
"time stamp": "2021-10-24 21:30:36.311",
"index": 5,
"Tx ": "Carol pays Donna 1 dscoin",
"nonce": 436,
"PrevHash": "0000088ABDDAD3B2658AA373BED8759D60C363B7ABED84D645B821FBD0DECEC9"
}
],
"chainHash": "005F0CB7E06498D4A5373506A7E383BCE9CC272E72FF54B8E90573AFC5D9F77B"
}
0. View basic blockchain status.
1. Add a transaction to the blockchain.

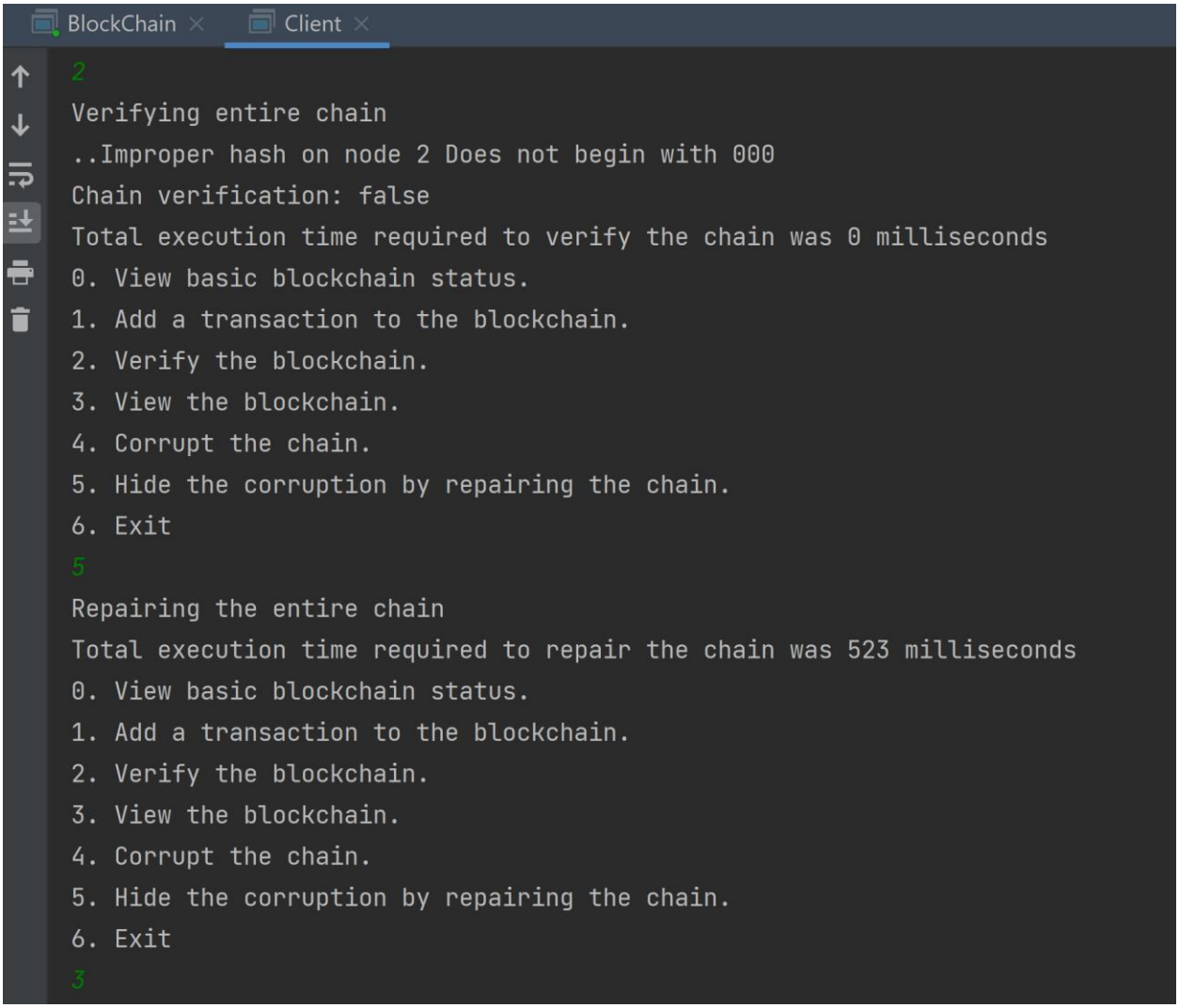

BlockChain
Client
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
5
Repairing the entire chain
Total execution time required to repair the chain was 299 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3
{
"ds_chain": [
{
"difficulty": 2,
"time stamp": "2021-10-24 21:27:49.517",
"index": 0,
"Tx ": "Carol pays Shivani 1000 dscoin",
"nonce": 591,

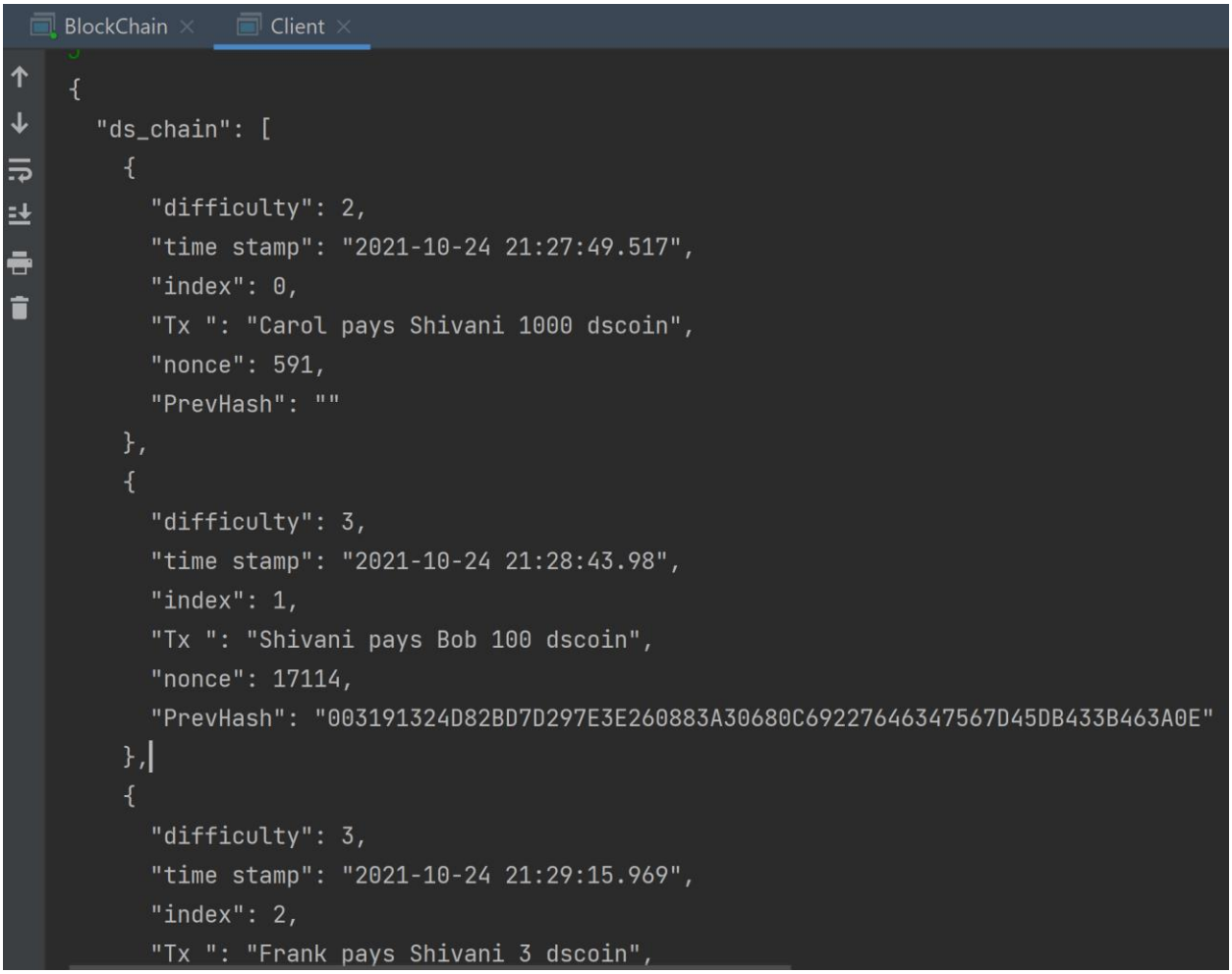

BlockChain
Client
"PrevHash": ""
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:28:43.98",
"index": 1,
"Tx ": "Shivani pays Bob 100 dscoin",
"nonce": 17114,
"PrevHash": "003191324D82BD7D297E3E260883A30680C692276463475 67D45DB433B463A0E"
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:29:15.969",
"index": 2,
"Tx ": "Bob pays Shivani 20 dscoin",
"nonce": 27393,
"PrevHash": "000BD63AA68788E5D07D16503C70070267BFBEC013766ABCD193A07C764232F8"
},
{
"difficulty": 4,
"time stamp": "2021-10-24 21:29:41.114",
"index": 3,
"Tx ": "Shivani pays Carol 23 dscoin",

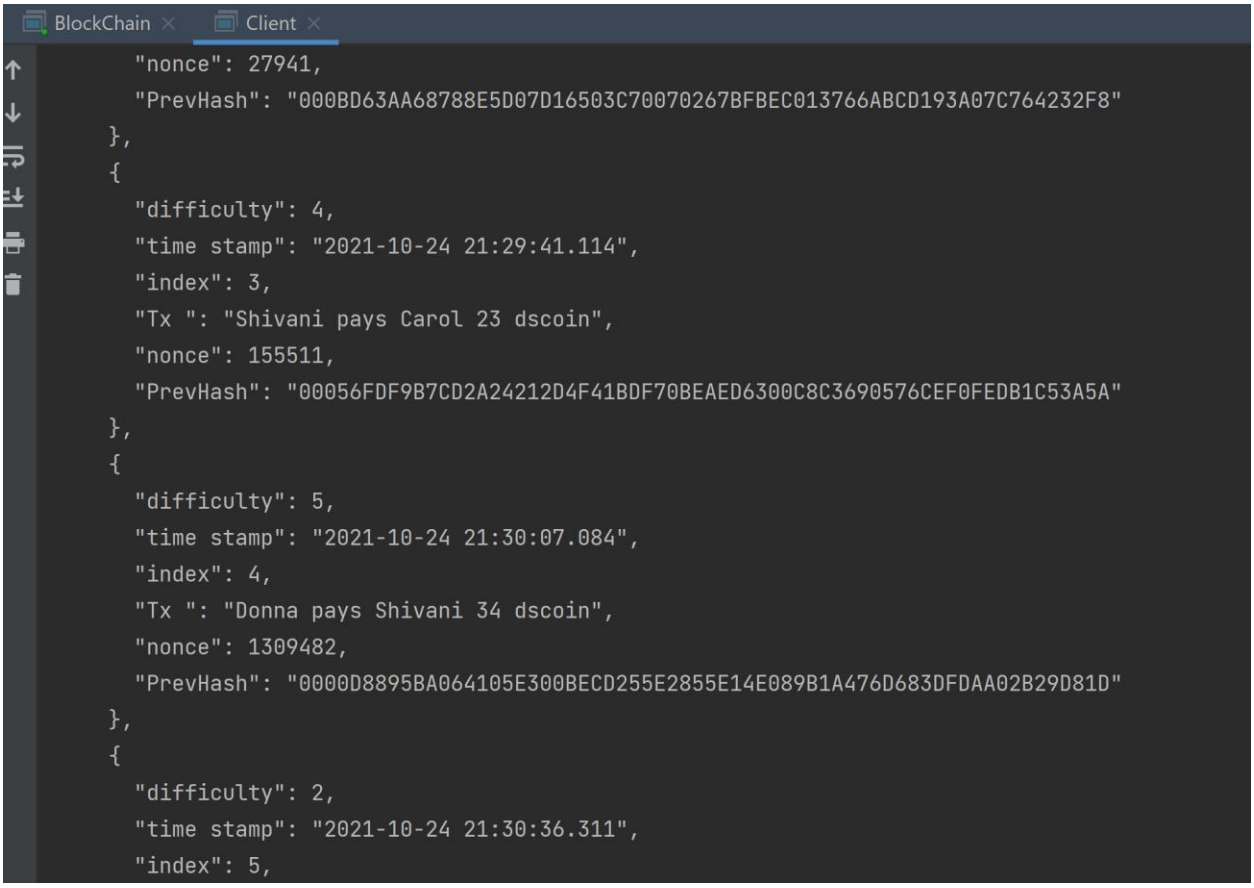

BlockChain
Client
"Tx ": "Shivani pays Carol 23 dscoin",
"nonce": 83529,
"PrevHash": "000E27035A059200519546C6FC91EBD8CF9677F4CF40BE3D9B1AB0E66BAD0859"
},
{
"difficulty": 5,
"time stamp": "2021-10-24 21:30:07.084",
"index": 4,
"Tx ": "Donna pays Shivani 34 dscoin",
"nonce": 725750,
"PrevHash": "0000A6A10CEBF9B2269CCAFD6C01902E5BFDBD05901BB0E07006FCF081FB9871"
},
{
"difficulty": 2,
"time stamp": "2021-10-24 21:30:36.311",
"index": 5,
"Tx ": "Carol pays Donna 1 dscoin",
"nonce": 1417,
"PrevHash": "0000087B06EAFBDDC4CCE3A8AFE8C1FAFDD032F0AC52C1A182C515068A2AB8E4"
}
],
"chainHash": "00B48401BBFFD034BF91213C691C69A52014172654AF044821A0EC024671CA4D"
}

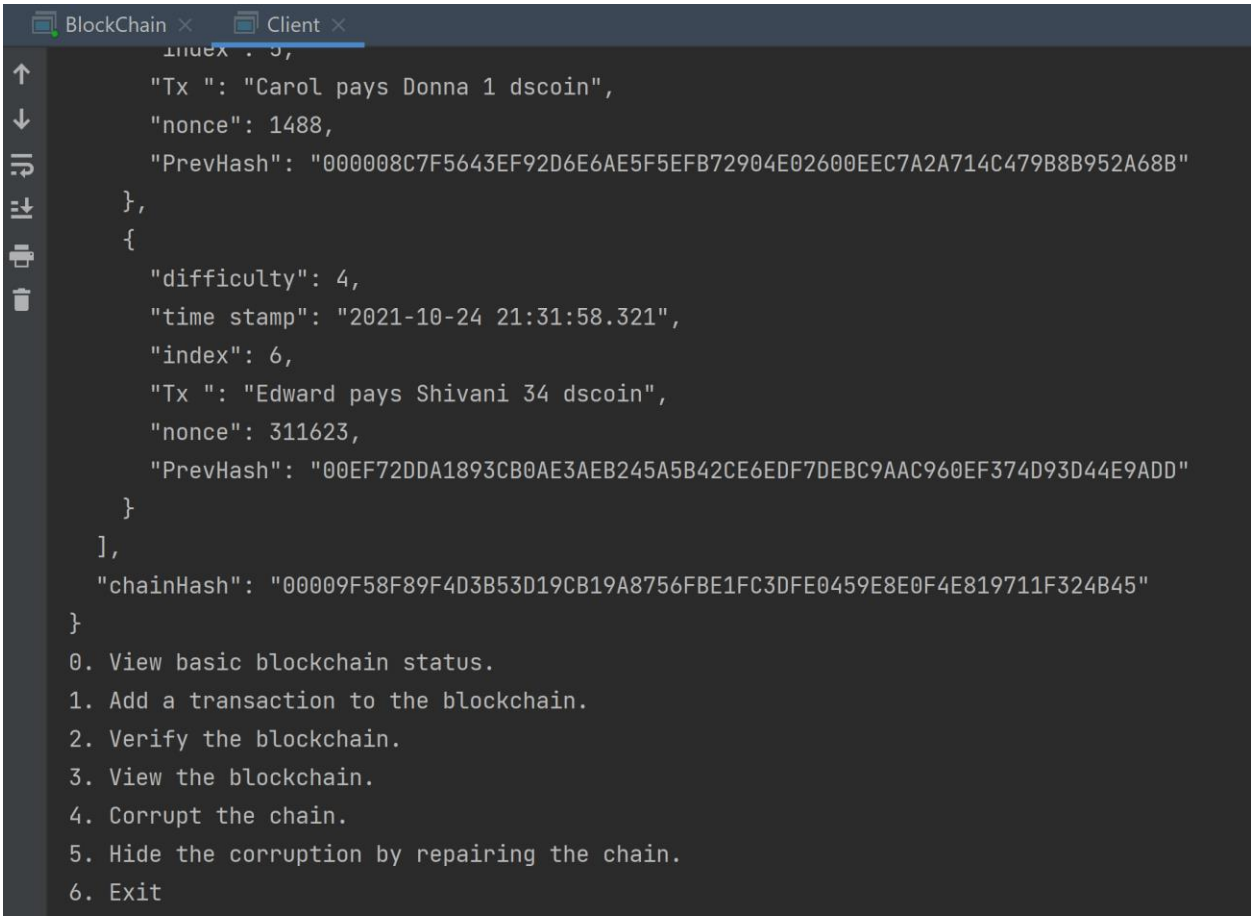

BlockChain
Client
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
1
Enter difficulty > 0
4
Enter transaction
Edward pays Shivani 34 dscoin
Total execution time to add this block was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
2
Verifying entire chain
Chain verification: true

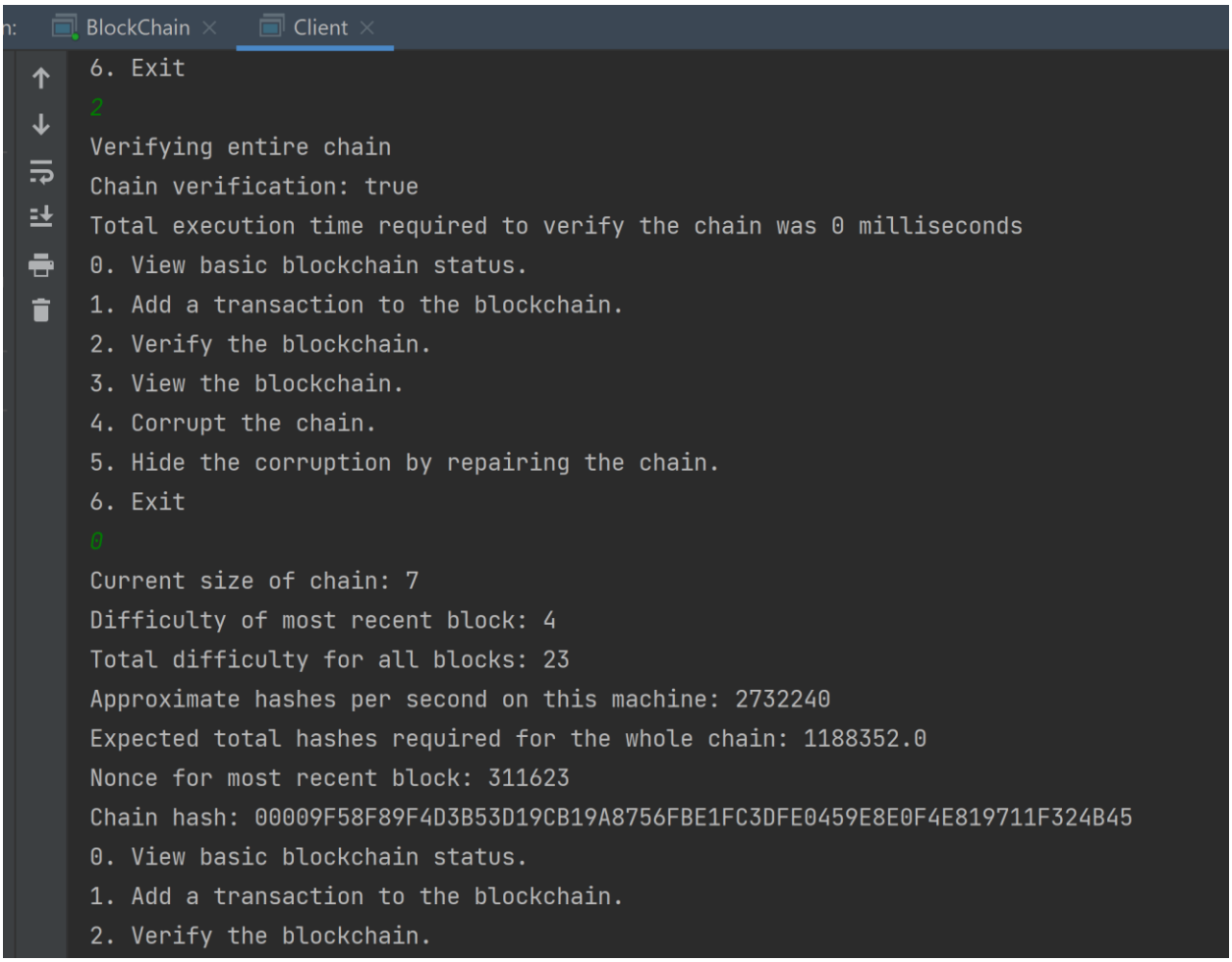

BlockChain
Client
Chain verification: true
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
4
corrupt the Blockchain
Enter block ID of block to corrupt
2
Enter new data for block 2
Frank pays Shivani 3 dscoin
Block 2 now holds Frank pays Shivani 3 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

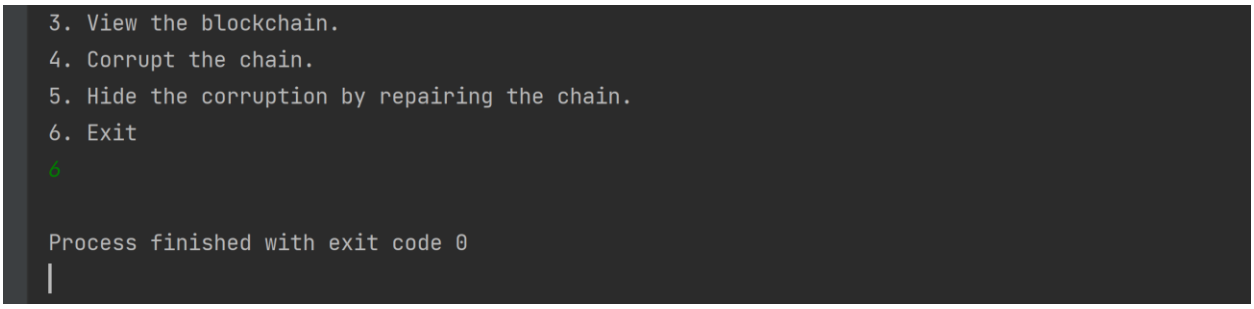

BlockChain
Client
2
Verifying entire chain
..Improper hash on node 2 Does not begin with 000
Chain verification: false
Total execution time required to verify the chain was 0 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
5
Repairing the entire chain
Total execution time required to repair the chain was 523 milliseconds
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit
3


BlockChain
Client
{
"ds_chain": [
{
"difficulty": 2,
"time stamp": "2021-10-24 21:27:49.517",
"index": 0,
"Tx ": "Carol pays Shivani 1000 dscoin",
"nonce": 591,
"PrevHash": ""
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:28:43.98",
"index": 1,
"Tx ": "Shivani pays Bob 100 dscoin",
"nonce": 17114,
"PrevHash": "003191324D82BD7D297E3E260883A30680C69227646347567D45DB433B463A0E"
},
{
"difficulty": 3,
"time stamp": "2021-10-24 21:29:15.969",
"index": 2,
"Tx ": "Frank pays Shivani 3 dscoin",


BlockChain
Client
"nonce": 27941,
"PrevHash": "000BD63AA68788E5D07D16503C70070267BFBEC013766ABCD193A07C764232F8"
},
{
"difficulty": 4,
"time stamp": "2021-10-24 21:29:41.114",
"index": 3,
"Tx ": "Shivani pays Carol 23 dscoin",
"nonce": 155511,
"PrevHash": "00056FDF9B7CD2A24212D4F41BDF70BEAED6300C8C3690576CEF0FEDB1C53A5A"
},
{
"difficulty": 5,
"time stamp": "2021-10-24 21:30:07.084",
"index": 4,
"Tx ": "Donna pays Shivani 34 dscoin",
"nonce": 1309482,
"PrevHash": "0000D8895BA064105E300BECD255E2855E14E089B1A476D683DFDAA02B29D81D"
},
{
"difficulty": 2,
"time stamp": "2021-10-24 21:30:36.311",
"index": 5,


BlockChain
Client
"Tx ": "Carol pays Donna 1 dscoin",
"nonce": 1488,
"PrevHash": "000008C7F5643EF92D6E6AE5F5EFB72904E02600EEC7A2A714C479B8B952A68B"
},
{
"difficulty": 4,
"time stamp": "2021-10-24 21:31:58.321",
"index": 6,
"Tx ": "Edward pays Shivani 34 dscoin",
"nonce": 311623,
"PrevHash": "00EF72DDA1893CB0AE3AEB245A5B42CE6EDF7DEBC9AAC960EF374D93D44E9ADD"
}
],
"chainHash": "00009F58F89F4D3B53D19CB19A8756FBE1FC3DFE0459E8E0F4E819711F324B45"
}
0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit





5. Task 1 Client Source Code

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 24th October 2021
 *
 * This code forms the client for the interaction with the BlockChain class
 * which serves as the server. The client takes in user input corresponding
 * to the user choice as in Task 0 and based on the user input, takes other
 * necessary details form the user and passed the data in JSON format
converted
```

```
 * to a string to the server. The GSON library and JSONObject libraries are
 * imported to help form the JSON Object and parse the JSON.
 * The client socket is initialized and forms a
 * connection with the server socket having port number 7777.The client
passes
 * the data entered by the user to the server via the socket connection
formed.
 * The operation logic happens in the server. The program runs till the user
chooses
 * option 6, i.e, exit. When the user enters 6 the connection is
terminated,however,
 * the server continues running.
 * */

import com.google.gson.*;
import org.json.simple.JSONObject;

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

public class Client {

    /*
    * Method takes user input and passes data to the getResult method
    * which interacts with the server and returns data as sent from
    * the server. The data is then printed according to needed format
    * using printOutput method. The process continues as long as user
    * does not enter 6, i.e, Exit
    * */
    public static void main(String [] args) {
        try {
            // stores user choice
            int userInput = -1;
            while (userInput != 6) {
                // displays menu options
                displayMenu();
                // Scanner to get user input from console
                Scanner getInput = new Scanner(System.in);
                userInput = Integer.parseInt(getInput.nextLine());
                // gets all necessary data from user and send to getResult
method
                // and getResult method returns data sent by server
                String output = getResult(getOperationData(userInput,
getInput));
                // to print data sent by server
                printOutput(output);
            }
        } catch (IOException e) {
            System.out.println("IO Exception: "+e.getMessage());
        } catch (Exception e) {
            System.out.println("Exception: "+e.getMessage());
        }
    }

    // this method parses the json string passed to print the output as
```

```
required
    // by parsing the contents of the json
    // Code from :
    // https://stackoverflow.com/questions/5490789/json-parsing-using-gson-
for-java
    private static void printOutput(String output) {
        // JSON parser
        JsonElement jsonElement = new JsonParser().parse(output);
        // creating JSONObject after parsing
        JsonObject jsonObject = jsonElement.getAsJsonObject();
        int userInput = jsonObject.get("user_input").getAsInt();
        switch (userInput) {
            case 0 -> {
                System.out.println("Current size of chain: " +
jsonObject.get("chain_size").getAsString());
                System.out.println("Difficulty of most recent block: " +
jsonObject.get("latest_difficulty").getAsString());
                System.out.println("Total difficulty for all blocks: " +
jsonObject.get("total_difficulty").getAsString());
                System.out.println("Approximate hashes per second on this
machine: " + jsonObject.get("hash_per_sec").getAsString());
                System.out.println("Expected total hashes required for the
whole chain: " + jsonObject.get("total_exp_hash").getAsString());
                System.out.println("Nonce for most recent block: " +
jsonObject.get("latest_nonce").getAsString());
                System.out.println("Chain hash: " +
jsonObject.get("chain_hash").getAsString());
            }
            case 1 -> System.out.println("Total execution time to add this
block was " + jsonObject.get("computation_time").getAsString() + "
milliseconds");
            case 2 -> {
                boolean isChainVerified
=jsonObject.get("chain_verified").getAsBoolean();
                if (!isChainVerified) {
                    int incorrectNode =
jsonObject.get("incorrect_node").getAsInt();
                    int difficulty = jsonObject.get("difficulty").getAsInt();
                    System.out.printf("..Improper hash on node %d Does not
begin with %s\n", incorrectNode, "0".repeat(Math.max(0, difficulty)));
                }
                System.out.println("Chain verification: " + isChainVerified);
                System.out.println("Total execution time required to verify
the chain was " + jsonObject.get("computation_time").getAsString() + "
milliseconds");
            }
            case 3 -> {
                // Code from
https://stackoverflow.com/questions/4105795/pretty-print-json-in-java
                // prettifying JSON to print content one below the other
                Gson gson = new GsonBuilder().setPrettyPrinting().create();
                String prettyJson = gson.toJson(new
JsonParser().parse(jsonObject.get("output_json").getAsString()));
                System.out.println(prettyJson);
            }
            case 4 -> {
                int blockId = jsonObject.get("block_id").getAsInt();
```

```
                String blockTransaction =
jsonObject.get("block_transaction").getAsString();
                System.out.printf("Block %d now holds %s\n", blockId,
blockTransaction);
            }
            case 5 -> System.out.printf("Total execution time required to
repair the chain was %s milliseconds\n",
jsonObject.get("computation_time").getAsString());
        }
    }

    /*
     * This method is used to create connection with the server and
     * pass the data to the server to perform the operation selected
     * by user and returned required data. The client Socket is initialized
with
     * source and destination. The BufferedReader in is used to read data
sent from the
     * server and the PrintWriter out is used to write into the socket. The
payload is written
     * into the socket using the PrintWriter out. On using out.flush() the
data in the stream
     * is sent to the server to perform the logic. The reply sent by the
server is read using
     * the BufferedReader in and returned to the calling method,i.e., the
main method
     * */
    public static String getResult(String payload) throws IOException {
        // Code from EchoClientTCP.java in Project 2
        // client socket declared
        Socket clientSocket = null;
        try {
            // server port number
            int serverPort = 7777;
            // client socket initialized
            clientSocket = new Socket("localhost", serverPort);
            // read data from socket
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            // write data to socket
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
            // write into socket
            out.println(payload);
            // send data written to server socket
            out.flush();
            // return data to main method
            return in.readLine();
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        } finally {
            // close socket connection
            if(clientSocket!=null) clientSocket.close();
        }
        return null;
    }
```

```java
    // this method gets other details from the user based on the menu option
selected.
    // The data collected is made into a JSON object and the JSON string is
then
    // returned to the calling method, i.e, the getResult method.
    // Code from https://stackoverflow.com/questions/5490789/json-parsing-
using-gson-for-java
    private static String getOperationData(int userInput,Scanner
getUserInput) {
        // new JSON Object created
        JSONObject json = new JSONObject();
        // add user input to JSON
        json.put("user_input",userInput);
        switch (userInput) {
            case 1 -> {
                System.out.println("Enter difficulty > 0");
                int difficulty = Integer.parseInt(getUserInput.nextLine());
                System.out.println("Enter transaction");
                String transaction = getUserInput.nextLine();
                // add difficulty to JSON object
                json.put("difficulty",difficulty);
                // add transaction to JSON object
                json.put("transaction",transaction);
            }
            case 2 -> System.out.println("Verifying entire chain");
            case 4 -> {
                System.out.println("corrupt the Blockchain\n" +
                        "Enter block ID of block to corrupt");
                int blockId = Integer.parseInt(getUserInput.nextLine());
                System.out.println("Enter new data for block " + blockId);
                String blockTransaction = getUserInput.nextLine();
                // add blockId to JSON object
                json.put("block_id",blockId);
                // add blockTransaction to JSON object
                json.put("block_transaction",blockTransaction);
            }
            case 5 -> System.out.println("Repairing the entire chain");
        }
        // return JSON string
        return json.toString();
    }

    // displays menu options to the user
    public static void displayMenu() {
        System.out.println("""
                0. View basic blockchain status.
                1. Add a transaction to the blockchain.
                2. Verify the blockchain.
                3. View the blockchain.
                4. Corrupt the chain.
                5. Hide the corruption by repairing the chain.
                6. Exit""");
    }
}
```

6. Task 1 Server Source Code

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 24th October 2021
 *
 * This code follows Task 0 but uses TCP for the data transmission between
the client
 * and server. The following code is the server side for the program.
 * The server socket is initialized and continues to listen to any
 * request sent by client sockets connected to port number 7777.The server
 * receives the data from the client through Scanner "in" via the socket
connection
 * formed. The PrintWriter "out" is used to write into the stream and send
data
 * back to the requesting client. The performOperations method checks the
operation
 * passed and does the required logic on the blockchain.The server is always
running.
 * Explanation and code taken from JavaDoc -
 * https://www.andrew.cmu.edu/course/95-
702/examples/javadoc/blockchaintask0/BlockChain.html#isChainValid()
 * This is the code for 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 previous hash and a difficulty of 2. On start up, this code will
also establish the hashes per
 * second instance member. All blocks added to the Blockchain will have a
difficulty passed by the
 * client. 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 three,
 * then all hashes will begin with 3 or more zero hex digits (or 3 nibbles,
or 12 zero bits).
 * It is menu-driven and based on the user selection, different data is sent
from the client and
 * certain operations are performed on the blockchain on the server side.
 * */

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import com.google.gson.*;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.*;
```

```
public class BlockChain {
    // ArrayList to hold Blocks
    private final ArrayList<Block> blocks;
    // holds a SHA256 hash of the most recently added Block.
    private String chainHash;
    // Store hashes per second
    private static int hashesPerSecond;
    // store node having wrong hash
    private int incorrectNode;

    // constructor
    BlockChain() {
        // initialize blocks ArrayList, chainHash and hashesPerSecond
        blocks= new ArrayList<>();
        chainHash = "";
        hashesPerSecond=0;
    }

    /*
     * On startup the genesis block is created and added to the blockchain.
Based on the data
     * sent from the client different operations are performed in the
performOperations method.
     *
     * Based on the experiments conducted the computation time increases as
the difficulty associated
     * with the block increases. For instance, for difficulty 2, addBlock()
methods takes between
     * 0-15 milliseconds approximately. For difficulty 5, addBlock() methods
took 461 milliseconds
     * approximately. For difficulty 6, addBlock() methods took 6000
milliseconds approximately.
     * The repairChain() and isChainValid methods take approximately 0
milliseconds if all the blocks
     * have correct data irrespective of the difficulty of the blocks.
However, in the presence of a
     * malicious block having corrupt data, repairChain() method takes longer
time if there are blocks
     * with higher difficulty. For instance, if malicious block has a
difficulty of 2, the
     * repairChain() method took 2775 milliseconds. But if malicious block
has a difficulty of 6, the
     * repairChain() method took 13221 milliseconds. isChainValid() method
also takes longer time
     * if malicious block has higher difficulty. For difficulty 2 of
malicious block, isChainValid()
     * method took close to 0 milliseconds and for difficulty 6, method took
close to 362 milliseconds.
     * */
    public static void main(String[] args){
        BlockChain blockChain = new BlockChain();
        Block genesisBlock = new Block(0, new Timestamp(new
Date().getTime()), "Genesis", 2);
        genesisBlock.setPreviousHash("");
        blockChain.computeHashesPerSecond();
        blockChain.addBlock(genesisBlock);
```

```
        // Code from EchoServerTCP.java in Project 2
        System.out.println("Server started");
        // client socket declared
        Socket clientSocket = null;
        try{
            int serverPort = 7777; // the server port number

            // Create a new server socket with port number 7777
            ServerSocket listenSocket = new ServerSocket(serverPort);

            // Since server is always running and listens for requests
            while(true){
                /*
                 * Block waiting for a new connection request from a client.
                 * When the request is received, "accept" it, and the rest
                 * the tcp protocol handshake will then take place, making
                 * the socket ready for reading and writing.
                 */
                if(clientSocket==null || clientSocket.getInputStream().read()
== -1)
                    clientSocket = listenSocket.accept();
                // If we get here, then we are now connected to a client.

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

                // Set up "out" to write to the client socket
                PrintWriter out;
                out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
                // read from client socket
                String data = in.nextLine();
                // result stores JSON string returned by performOperations
method
                String result = blockChain.performOperations(data);
                // write to client socket
                out.println(result);
                // send data written to client socket
                out.flush();
            }
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }
    }

    // check the proofOfWork for each block and add it to the blockchain,
update
    // chain hash with the latest added block's proofOfWork
    public void addBlock (Block newBlock) {
        if(blocks.size()!=0) newBlock.setPreviousHash(getBlock(blocks.size()-
1).proofOfWork());
        blocks.add(newBlock);
```

```
        chainHash = newBlock.proofOfWork();
    }

    // getter for timestamp
    public Timestamp getTime() {
        return new Timestamp(new Date().getTime());
    }

    // getter for latest block
    public Block getLatestBlock() {
        return getBlock(blocks.size()-1);
    }

    // getter for chain size
    public int getChainSize() {
        return blocks.size();
    }

    /*
     * This method computes exactly 1 million hashes and times how long that
process takes.
     * So, hashes per second is approximated as (1 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() {
        String sampleString = "00000000";
        long startTime = getTime().getTime();
        byte[] bytesOfHash = sampleString.getBytes(StandardCharsets.UTF_8);
        for(int i=0; i<1000000;i++) {
            try {
                // Use SHA-256 for hashing
                MessageDigest md = MessageDigest.getInstance("SHA-256");
                md.digest(bytesOfHash);
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
        }
        long endTime = getTime().getTime();
        BlockChain.hashesPerSecond = (int) (1000000*1000/ (double) (endTime -
startTime));
    }

    // getter for hashes per second
    public int getHashesPerSecond() {
        return hashesPerSecond;
    }

    // creates string in json format for entire blockchain using toString
method of
    // individual blocks
    public String toString() {
        System.out.println("View the Blockchain");
        JSONArray jsonBlocks = new JSONArray();
        for(int i=0; i<blocks.size();i++) {
            Block block = getBlock(i);
```

```
            JsonElement jsonElement = new
JsonParser().parse(block.toString());
            jsonBlocks.add(jsonElement);
        }
        JSONObject outputJson = new JSONObject();
        outputJson.put("ds_chain",jsonBlocks);
        outputJson.put("chainHash",chainHash);
        return outputJson.toString();
    }

    // getter for block at index i
    public Block getBlock (int i) {
        return blocks.get(i);
    }

    // compute and return the total difficulty of all blocks on the chain.
    public int getTotalDifficulty() {
        int totalDifficulty=0;
        for(int i=0; i< blocks.size(); i++) {
            totalDifficulty+=getBlock(i).getDifficulty();
        }
        return totalDifficulty;
    }

    // Compute and return the expected number of hashes required for the
entire chain.
    public double getTotalExpectedHashes() {
        double totalExpectedHashes = 0;
        for(int i=0; i<blocks.size();i++) {
            totalExpectedHashes +=
Math.pow((16),getBlock(i).getDifficulty());
        }
        return totalExpectedHashes;
    }

    /*
     * If the chain only contains one block, the genesis block at position 0,
this method 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 false.
Otherwise, return true.
     * If the chain has more blocks than one, begin checking all blocks using
chainValidation
     * method. If chainValidation method returns a non-negative number
continue checking the
     * blocks in the blockchain. When chainValidation method returns -1, then
it means the
     * chain validation fails. */
    public boolean isChainValid() {
        if(blocks.size()==1) {
            Block genesis = getBlock(0);
            String genesisHash = genesis.proofOfWork();
            int numberOfZero = 0;
            int difficulty = genesis.getDifficulty();
```

```
            char[] charsHash = genesisHash.toCharArray();
            for (int i = 0; i < difficulty; i++) {
                if (charsHash[i] == '0') numberOfZero++;
            }
            return chainHash.equals(genesisHash) && numberOfZero ==
difficulty;
        } else {
            for(int i=0; i<blocks.size();i++) {
                // checks each node
                int node = chainValidation(i);
                if(node==-1) {
                    // setting index of incorrect node with wrong hash value
                    incorrectNode=i-1;
                    return false;
                }
            }
            return chainHash.equals(getLatestBlock().proofOfWork());
        }
    }

    /*
     * Check hash pointer of each Block and compare with proofOfWork of
previous block.
     * If they match and if the proof of work is correct, return block index.
else
     * return -1; */
    public int chainValidation(int blockIndex) {
        String prevBlockHash = blockIndex==0?
                "":getBlock(blockIndex-1).proofOfWork();
        String hashPointer = getBlock(blockIndex).getPreviousHash();
        if(prevBlockHash.equals(hashPointer)) {
            if(blockIndex!=0){
                char[] charsHash = prevBlockHash.toCharArray();
                int difficulty = getBlock(blockIndex-1).getDifficulty();
                int numberOfZero = 0;
                for (int i = 0; i < difficulty; i++) {
                    if (charsHash[i] == '0') numberOfZero++;
                }
                if (numberOfZero == difficulty) return blockIndex;
            }
            return blockIndex;
        }
        return -1;
    }

    // checks each block and recomputes proofOfWork for each block and
    // assigns the correct hash pointer in the next block as well as the
    // chain hash using the proofOfWork of the last added block.
    public void repairChain() {
        for(int i=0; i<getChainSize();i++) {
            if(i==0) getBlock(i).setPreviousHash("");
            String correctHash = getBlock(i).proofOfWork();
            if(i+1 <
getChainSize())getBlock(i+1).setPreviousHash(correctHash);
            if(i==getChainSize()-1) chainHash=correctHash;
        }
        chainHash=getLatestBlock().proofOfWork();
```

```
    }

    // this method parses the json string sent from the client and
    // performs the required operation by parsing through the
    // json object. Further, it creates a new outputPayload
    // json object,converts the json object to string inorder
    // to send back to the requesting client
    public String performOperations(String data) {
        // JSON Parsing
        JsonElement jsonElement = new JsonParser().parse(data);
        // Creating JSON object from parsed string
        JsonObject jsonObject = jsonElement.getAsJsonObject();
        int userInput = jsonObject.get("user_input").getAsInt();
        // JSON object to be returned to client
        JSONObject outputPayload = new JSONObject();
        // load output JSON with user input
        outputPayload.put("user_input",userInput);
        switch (userInput) {
            case 0 -> {
                // loads json with basic blockchain status
                outputPayload.put("chain_size",getChainSize());

outputPayload.put("latest_difficulty",getLatestBlock().getDifficulty());
                outputPayload.put("total_difficulty",getTotalDifficulty());
                outputPayload.put("hash_per_sec",getHashesPerSecond());
                outputPayload.put("total_exp_hash",getTotalExpectedHashes());

outputPayload.put("latest_nonce",getLatestBlock().getNonce());
                outputPayload.put("chain_hash",chainHash);
            }
            case 1 -> {
                // load output JSON with computation time after adding new
block
                long start = System.currentTimeMillis();
                int difficulty
=Integer.parseInt(jsonObject.get("difficulty").getAsString());
                String transaction =
jsonObject.get("transaction").getAsString();
                addBlock(new Block(getLatestBlock().getIndex() + 1,
                        new Timestamp(new Date().getTime()), transaction,
difficulty));
                long end = System.currentTimeMillis();
                outputPayload.put("computation_time",end-start);
            }
            case 2 -> {
                // performs chain verification, loads output JSON with
incorrectNode id
                // and difficulty in case chain verification fails. load
output JSON with
                // computation time for the process as well
                long start = System.currentTimeMillis();
                boolean chainVerified = isChainValid();
                outputPayload.put("chain_verified",chainVerified);
                if (!chainVerified) {
                    int difficulty = getBlock(incorrectNode).getDifficulty();
                    outputPayload.put("incorrect_node",incorrectNode);
                    outputPayload.put("difficulty",difficulty);
```

```
                }
                long end = System.currentTimeMillis();
                outputPayload.put("computation_time",end-start);
            }
            // loads output json with blockchain content
            case 3 ->  outputPayload.put("output_json",this.toString());
            case 4 -> {
                // updates block data based on block id and transaction
passed from
                // the client side and loads output json with the block id
and transaction
                int blockId =
Integer.parseInt(jsonObject.get("block_id").getAsString());
                String blockTransaction
=jsonObject.get("block_transaction").getAsString();
                getBlock(blockId).setData(blockTransaction);
                outputPayload.put("block_id",blockId);
                outputPayload.put("block_transaction",blockTransaction);
            }
            case 5 -> {
                // repairs chain and loads output json with computation time
needed
                // for this process
                long start = System.currentTimeMillis();
                repairChain();
                long end = System.currentTimeMillis();
                outputPayload.put("computation_time",end-start);
            }
        }
        // return json string
        return outputPayload.toString();
    }
}
```