***Project 3***

***Ruth Peter***

***Andrew id : rpeter***

***Task 0 Execution***

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=51761:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ruthpeter/ds/Project3/untitled/target/classes:/Users/ruthpeter/.m2/repository/com/google/code/gson/gson/2.2.4/gson-2.2.4.jar:/Users/ruthpeter/.m2/repository/org/json/json/20230227/json-20230227.jar blockchaintask0.BlockChain

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Approximate hashes per second on this machine : 2000000

Expected total hashes required for the whole chain: 256.0

Nonce for most recent block: 360

Chain hash: 00e79eeeb4dd22af6de5864b52f87689b10bd5c4415d95b4dadb0ef32dfaf053

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Alice pays Bill 100 DSCoin

Total execution time to add this block was 6 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Bill pays Clara 50 DSCoin

Total execution time to add this block was 3 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Clara pays Daisy 10 DS Coin

Total execution time to add this block was 6 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

2

Chain verification: TRUE

Total execution time to verify the chain was 3 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

3

View the Blockchain

[{"index":0,"timestamp":"2023-03-17 02:58:50.708","tx":"Genesis","previousHash":"","nonce":360,"difficulty":2},{"index":1,"timestamp":"2023-03-17 02:59:11.775","tx":"Alice pays Bill 100 DSCoin","previousHash":"00e79eeeb4dd22af6de5864b52f87689b10bd5c4415d95b4dadb0ef32dfaf053","nonce":465,"difficulty":2},{"index":2,"timestamp":"2023-03-17 02:59:29.133","tx":"Bill pays Clara 50 DSCoin","previousHash":"00159581265aa9d1e84802bca1e33337bb39952e39e44577a7678476038ca536","nonce":55,"difficulty":2},{"index":3,"timestamp":"2023-03-17 02:59:48.485","tx":"Clara pays Daisy 10 DS Coin","previousHash":"0016603b56922e6d7ea09b6f1998e2f945751684b4e46005c6abc7287835daa6","nonce":232,"difficulty":2}]

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

4

corrupt the Blockchain

Enter block ID of block to corrupt

1

Enter new data for block 1

Alice pays Bill 76 DSCoin

Block 1 now holds Alice pays Bill 76 DSCoin

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

3

View the Blockchain

[{"index":0,"timestamp":"2023-03-17 02:58:50.708","tx":"Genesis","previousHash":"","nonce":360,"difficulty":2},{"index":1,"timestamp":"2023-03-17 02:59:11.775","tx":"Alice pays Bill 76 DSCoin","previousHash":"00e79eeeb4dd22af6de5864b52f87689b10bd5c4415d95b4dadb0ef32dfaf053","nonce":465,"difficulty":2},{"index":2,"timestamp":"2023-03-17 02:59:29.133","tx":"Bill pays Clara 50 DSCoin","previousHash":"00159581265aa9d1e84802bca1e33337bb39952e39e44577a7678476038ca536","nonce":55,"difficulty":2},{"index":3,"timestamp":"2023-03-17 02:59:48.485","tx":"Clara pays Daisy 10 DS Coin","previousHash":"0016603b56922e6d7ea09b6f1998e2f945751684b4e46005c6abc7287835daa6","nonce":232,"difficulty":2}]

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

2

Chain verification: FALSE

Improper hash on node 1 Does not begin with 00

Total execution time to verify the chain was 3 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

5

Total execution time required to repair the chain was 9 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

2

Chain verification: TRUE

Total execution time to verify the chain was 1 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

4

Enter transaction:

Daisy pays Sean 25 DSCoin

Total execution time to add this block was 13 milliseconds

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

0

Current size of chain: 5

Difficulty of most recent block: 4

Total difficulty for all blocks: 12

Approximate hashes per second on this machine : 2000000

Expected total hashes required for the whole chain: 66560.0

Nonce for most recent block: 2108

Chain hash: 00002468a01dd3ae0cd46cf86ca9ed68a220217a0c94992a87e9018036d52d09

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

6

Process finished with exit code 0

***Task 0 Block.java***

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*/*package blockchaintask0;  
  
import com.google.gson.JsonObject;  
  
import java.math.BigInteger;  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.sql.Timestamp;  
  
public class Block {  
  
 // the position of the block on the chain  
 private int index;  
  
 //time of the block's creation  
 private Timestamp timestamp;  
  
 //block's single transaction details  
 private String data;  
  
 //the SHA256 hash of a block's parent  
 private String previousHash;  
  
 //determined by POW routine  
 private BigInteger nonce;  
  
 //minimum number of left most hex digits needed by a proper hash  
 private int difficulty;  
  
 public Block(int index, Timestamp timestamp, String data, int difficulty) {  
 this.index = index;  
 this.timestamp = timestamp;  
 this.data = data;  
 this.difficulty = difficulty;  
 this.nonce = BigInteger.*ZERO*;  
 }  
  
  
 */\*\*  
 \* This method computes a hash of the concatenation of the index,  
 \* timestamp, data, previousHash, nonce, and difficulty.  
 \*  
 \** ***@return*** *a String holding Hexadecimal characters  
 \*/* public String calculateHash() throws NoSuchAlgorithmException {  
 String parentString = String.*valueOf*(this.index) + this.timestamp + this.data + this.previousHash + this.nonce + this.difficulty;  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 byte[] encodedHash = md.digest(  
 parentString.getBytes(StandardCharsets.*UTF\_8*));  
 return BlockHelper.*bytesToHex*(encodedHash);  
 }  
  
  
 */\*\*  
 \* This method returns the nonce for this block  
 \*  
 \** ***@return*** *a BigInteger representing the nonce for this block  
 \*/* public BigInteger getNonce() {  
 return nonce;  
 }  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *difficulty  
 \*/* public int getDifficulty() {  
 return difficulty;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *difficulty - determines how much work is required to produce a proper hash  
 \*/* public void setDifficulty(int difficulty) {  
 this.difficulty = difficulty;  
 }  
  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *index  
 \*/* public int getIndex() {  
 return index;  
 }  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *timestamp of this block  
 \*/* public Timestamp getTimestamp() {  
 return timestamp;  
 }  
  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *this block's transaction  
 \*/* public String getData() {  
 return data;  
 }  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *previous hash  
 \*/* public String getPreviousHash() {  
 return previousHash;  
 }  
  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *index - the index of this block in the chain  
 \*/* public void setIndex(int index) {  
 this.index = index;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *timestamp - of when this block was created  
 \*/* public void setTimestamp(Timestamp timestamp) {  
 this.timestamp = timestamp;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *data - represents the transaction held by this block  
 \*/* public void setData(String data) {  
 this.data = data;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *previousHash - a hashpointer to this block's parent  
 \*/* public void setPreviousHash(String previousHash) {  
 this.previousHash = previousHash;  
 }  
  
  
 */\*\*  
 \** ***@return*** *A JSON representation of all of this block's data is returned  
 \*/* @Override  
 public String toString() {  
  
 JsonObject jsonObject = new JsonObject();  
  
 jsonObject.addProperty("index", index);  
 jsonObject.addProperty("timestamp", String.*valueOf*(timestamp));  
 jsonObject.addProperty("tx", data);  
 jsonObject.addProperty("previousHash", previousHash);  
 jsonObject.addProperty("nonce", nonce);  
 jsonObject.addProperty("difficulty", difficulty);  
 return jsonObject.toString();  
 }  
  
 */\*\*  
 \* The proof of work methods finds a good hash.  
 \*  
 \** ***@return*** *a String with a hash that has the appropriate number of leading hex zeroes.  
 \*/*public String proofOfWork() throws NoSuchAlgorithmException {  
  
 String hexHash = calculateHash();  
  
 String matchString = "";  
 //this.difficulty  
  
 for (int i = 0; i < this.difficulty; i++) {  
 matchString = matchString + "0";  
 }  
  
 while(!hexHash.substring(0, this.difficulty).equalsIgnoreCase(matchString)){  
 this.nonce = this.nonce.add(BigInteger.*ONE*);  
 hexHash = calculateHash();  
 }  
  
 return hexHash;  
  
}

***Task 0 BlockChain.java***

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*/*package blockchaintask0;  
  
import com.google.gson.Gson;  
import com.google.gson.JsonArray;  
import com.google.gson.JsonElement;  
  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.sql.Timestamp;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class BlockChain {  
  
 private List<Block> blockList;  
  
 private String chainHash;  
  
 private int hashesPerSecond;  
  
 Gson gson = new Gson();  
  
 public BlockChain() {  
 this.blockList = new ArrayList<>();  
 this.chainHash = "";  
 this.hashesPerSecond = 0;  
  
 }  
  
 */\*\*  
 \** ***@return*** *the chain hash  
 \*/* public String getChainHash() {  
 return chainHash;  
 }  
  
 */\*\*  
 \** ***@return*** *the current system time  
 \*/* public Timestamp getTime() {  
 return new Timestamp(System.*currentTimeMillis*());  
 }  
  
 */\*\*  
 \* a reference to the most recently added Block  
 \*  
 \** ***@return*** *\*/* public Block getLatestBlock() {  
 return blockList.get(blockList.size() - 1);  
 }  
  
 public int getChainSize() {  
 return blockList.size();  
 }  
  
 */\*\*  
 \* This method computes exactly 2 million hashes and times how long that process takes  
 \*/* public void computeHashesPerSecond() throws NoSuchAlgorithmException {  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 int i = 0;  
 Timestamp startTime = getTime();  
 while (i < 2000000) {  
 byte[] encodedHash = md.digest(  
 "00000000".getBytes(StandardCharsets.*UTF\_8*));  
 i += 1;  
 }  
 Timestamp endTime = getTime();  
 this.hashesPerSecond = 2000000 / (endTime.compareTo(startTime));  
  
 }  
  
 */\*\*  
 \* get hashes per second  
 \*  
 \** ***@return*** *the instance variable approximating the number of hashes per second  
 \*/* public int getHashesPerSecond() {  
 return this.hashesPerSecond;  
 }  
  
 */\*\*  
 \* Method to add new block to blockChain  
 \* block's previous hash must hold the hash of the most recently added block  
 \* the new block becomes the most recently added block on the BlockChain  
 \* The SHA256 hash of every block must exhibit proof of work  
 \*  
 \** ***@param*** *newBlock  
 \** ***@throws*** *NoSuchAlgorithmException  
 \*/* public void addBlock(Block newBlock) throws NoSuchAlgorithmException {  
 if (this.blockList.size() > 0)  
 newBlock.setPreviousHash(getChainHash());  
 else  
 newBlock.setPreviousHash("");  
 blockList.add(newBlock);  
 this.chainHash = newBlock.proofOfWork();  
 }  
  
  
 */\*\*  
 \* For difficulty 2  
 \* Time to add block = 5ms  
 \* Verify blockchain = 1ms  
 \* Repair chain = 5ms  
 \*  
 \* For difficulty 3  
 \* Time to add block = 24ms  
 \* Verify blockchain = 1ms  
 \* Repair chain = 1ms  
 \*  
 \* For difficulty 4  
 \* Time to add block = 51ms  
 \* Verify blockchain = 1ms  
 \* Repair chain = 6ms  
 \*  
 \*  
 \* As the difficulty of the block increases, the time taken to add block also increases exponentially  
 \* Verification of the block still approximately takes the same time  
 \* Repairing the block chain also approximately takes the same time  
 \*  
 \* The maximum time is taken during adding the block as it computes the proof of work for the difficulty given ,  
 \* and keeps increasing nonce till an appropriate hash is found , the higher the difficulty the more computations are  
 \* required  
 \*  
 \*  
 \*/* public static void main(String args[]) throws NoSuchAlgorithmException {  
  
 BlockChain chain = new BlockChain();  
  
 //adding genesis block to chain  
  
 Block genesis = new Block(0, chain.getTime(), "Genesis", 2);  
 //first block , previous hash is blank  
 genesis.setPreviousHash("");  
 //calculate pow  
 genesis.proofOfWork();  
  
 chain.computeHashesPerSecond();  
  
 //add 1st block to the main chain  
 chain.addBlock(genesis);  
  
  
 Scanner sc = new Scanner(System.*in*);  
 int option = 0;  
  
 while (true) {  
 System.*out*.println("0. View basic blockchain status.\n" +  
 "1. Add a transaction to the blockchain.\n" +  
 "2. Verify the blockchain.\n" +  
 "3. View the blockchain.\n" +  
 "4. Corrupt the chain.\n" +  
 "5. Hide the corruption by repairing the chain.\n" +  
 "6. Exit");  
  
 option = sc.nextInt();  
  
 switch (option) {  
 //Status of blockchain  
 case 0: {  
 System.*out*.println("Current size of chain: " + chain.getChainSize());  
 System.*out*.println("Difficulty of most recent block: " + chain.getLatestBlock().getDifficulty());  
 System.*out*.println("Total difficulty for all blocks: " + chain.getTotalDifficulty());  
 System.*out*.println("Approximate hashes per second on this machine : " + chain.getHashesPerSecond());  
 System.*out*.println("Expected total hashes required for the whole chain: " + chain.getTotalExpectedHashes());  
 System.*out*.println("Nonce for most recent block: " + chain.getLatestBlock().getNonce());  
 System.*out*.println("Chain hash: " + chain.getChainHash() + "\n");  
 break;  
 }  
  
 // adding block to blockchain  
 case 1: {  
 System.*out*.println("Enter difficulty > 0 \n");  
 int difficulty = sc.nextInt();  
 sc.nextLine();  
 System.*out*.println("Enter transaction: \n");  
 String transaction = sc.nextLine();  
  
 Timestamp start = chain.getTime();  
  
 //add new block to the chain  
 Block newBlock = new Block(chain.getChainSize(), chain.getTime(), transaction, difficulty);  
 newBlock.setPreviousHash(chain.getChainHash());  
 newBlock.proofOfWork();  
  
 chain.addBlock(newBlock);  
  
 Timestamp end = chain.getTime();  
  
  
 System.*out*.println("Total execution time to add this block was " + (end.getTime() - start.getTime()) + " milliseconds");  
 break;  
  
 }  
  
 //verify blockchain  
 case 2: {  
 Timestamp start = chain.getTime();  
 System.*out*.println("Chain verification: " + chain.isChainValid());  
 Timestamp end = chain.getTime();  
  
 System.*out*.println("Total execution time to verify the chain was " + (end.getTime() - start.getTime()) + " milliseconds");  
 break;  
 }  
  
 //view blockchain  
 case 3: {  
 System.*out*.println("View the Blockchain");  
 System.*out*.println(chain.toString());  
 break;  
  
 }  
  
 //corrupt blockchain  
 case 4: {  
 System.*out*.println("corrupt the Blockchain");  
 System.*out*.println("Enter block ID of block to corrupt");  
 int index = sc.nextInt();  
 sc.nextLine();  
 System.*out*.println("Enter new data for block " + index);  
 String transaction = sc.nextLine();  
 //setting data to the selected block in the chain  
 chain.getBlock(index).setData(transaction);  
 System.*out*.println("Block " + index + " now holds " + transaction);  
 break;  
 }  
  
 // repair blockchain  
 case 5: {  
 Timestamp start = chain.getTime();  
 chain.repairChain();  
 Timestamp end = chain.getTime();  
  
 System.*out*.println("Total execution time required to repair the chain was " + (end.getTime() - start.getTime()) + " milliseconds");  
 break;  
 }  
  
 //exit the program  
 case 6:  
 System.*exit*(0);  
 }  
 }  
 }  
  
 */\*\*  
 \* This method uses the toString method defined on each individual block  
 \*  
 \** ***@return*** *a String representation of the entire chain is returned  
 \*/* @Override  
 public String toString() {  
  
 JsonArray jsonArray = new JsonArray();  
  
 for (Block b : blockList) {  
 jsonArray.add(gson.fromJson(b.toString(), JsonElement.class));  
 }  
  
 return jsonArray.toString();  
  
 }  
  
 */\*\*  
 \* return block at position i  
 \*  
 \** ***@param*** *i  
 \** ***@return*** *\*/* public Block getBlock(int i) {  
 return blockList.get(i);  
 }  
  
  
 */\*\*  
 \* Compute and return the total difficulty of all blocks on the chain. Each block knows its own difficulty  
 \*  
 \** ***@return*** *totalDifficulty  
 \*/* public int getTotalDifficulty() {  
 int totalDifficulty = 0;  
 for (Block block : blockList) {  
 totalDifficulty += block.getDifficulty();  
 }  
 return totalDifficulty;  
 }  
  
 */\*\*  
 \* Compute and return the expected number of hashes required for the entire chain.  
 \*  
 \** ***@return*** *totalExpectedHashes  
 \*/* public double getTotalExpectedHashes() {  
 double totalExpectedHashes = 0;  
 for (Block block : blockList)  
 totalExpectedHashes += Math.*pow*(16, block.getDifficulty()); // 16 (16 hex characters) ^ difficulty of block  
 return totalExpectedHashes;  
 }  
  
 */\*\*  
 \* checks if the hash has requisite number of leftmost zeroes as specified in difficulty for the entire chain  
 \*  
 \** ***@return*** *"TRUE" if the chain is valid, otherwise return a string with an appropriate error message  
 \** ***@throws*** *NoSuchAlgorithmException  
 \*/* public String isChainValid() throws NoSuchAlgorithmException {  
 //chain contains only 1 block , i.e. genesis  
 if (blockList.size() == 1) {  
 Block genesisBlock = this.blockList.get(0);  
 String hash = genesisBlock.calculateHash();  
 //calculate prefix based on difficulty, number of leading zeroes based on the difficulty value  
 String prefix = new String(new char[genesisBlock.getDifficulty()]).replace("\0", "0");  
 if (!hash.substring(0, genesisBlock.getDifficulty()).equals(prefix)) {  
 return "FALSE \n Improper hash on genesis node";  
 } else if (!chainHash.equals(hash)) {  
 return "FALSE \n Chain hash and computed hash do not match";  
 } else {  
 return "TRUE";  
 }  
 }  
  
 //more than 1 block  
 if (blockList.size() > 1) {  
 for (int i = 1; i < blockList.size(); i++) {  
  
 Block currentBlock = this.blockList.get(i);  
 Block previousBlock = this.blockList.get(i - 1);  
  
  
 String hash = currentBlock.calculateHash();  
 String hashPointer = currentBlock.getPreviousHash();  
 //calculate prefix based on difficulty, number of leading zeroes based on the difficulty value  
 String prefix = new String(new char[currentBlock.getDifficulty()]).replace("\0", "0");  
  
 if (!hash.substring(0, currentBlock.getDifficulty()).equals(prefix))  
 return "FALSE \n Improper hash on node " + i + " Does not begin with " + prefix;  
 //check proof of work / leading zeros  
 else if (!hashPointer.equals(previousBlock.calculateHash()))  
 return "FALSE \n Improper previous hash";  
 }  
 }  
  
 //chain hash , check the last element added to to the blocklist  
 if (!chainHash.equals(blockList.get(blockList.size() - 1).calculateHash())) {  
 return "Chain hash error";  
 }  
  
 return "TRUE";  
 }  
  
 */\*\*  
 \* This routine repairs the chain. It checks the hashes of each block and ensures that any illegal hashes are recomputed.  
 \* After this routine is run, the chain will be valid. The routine does not modify any difficulty values.  
 \* It computes new proof of work based on the difficulty specified in the Block.  
 \*  
 \** ***@throws*** *NoSuchAlgorithmException  
 \*/* public void repairChain() throws NoSuchAlgorithmException {  
  
 //genesis block  
 if (blockList.size() == 1) {  
 //Reset previous hash and recompute proof of work  
 blockList.get(0).setPreviousHash("");  
 blockList.get(0).proofOfWork();  
 }  
  
  
 if (blockList.size() > 1) {  
 for (int i = 1; i < blockList.size(); i++) {  
 //reset previous hash and recompute proof of work  
 blockList.get(i).setPreviousHash(blockList.get(i - 1).calculateHash());  
 blockList.get(i).proofOfWork();  
 }  
  
 //reset chain hash  
 this.chainHash = blockList.get(blockList.size() - 1).calculateHash();  
 }  
 }  
  
}

***Task 1 Client Side Execution***

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=61152:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ruthpeter/ds/Project3/Project3Task1/target/classes:/Users/ruthpeter/.m2/repository/com/google/code/gson/gson/2.2.4/gson-2.2.4.jar:/Users/ruthpeter/.m2/repository/org/json/json/20230227/json-20230227.jar blockchaintask1.EchoClientTCP

The TCP client is running.

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

0

Current size of chain: 1

Difficulty of most recent block: 2

Total difficulty for all blocks: 2

Approximate hashes per second on this machine : 2000000

Expected total hashes required for the whole chain: 256.0

Nonce for most recent block: 212

Chain hash: "000e489c7e46345eabd8486f748a69e64f3cdc956ae6b5d2faa82b935c9b5a6b"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Alice pays Bill 100 DSCoin

"Total execution time to add this block was 7 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Bill pays Clara 50 DSCoin

"Total execution time to add this block was 3 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Clara pays Daisy 10 DS Coin

"Total execution time to add this block was 6 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

2

"Total execution time to verify the chain was 1 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

3

"[{\"index\":0,\"timestamp\":\"2023-03-17 13:51:48.715\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":212,\"difficulty\":2},{\"index\":1,\"timestamp\":\"2023-03-17 13:52:10.226\",\"tx\":\"Alice pays Bill 100 DSCoin\",\"previousHash\":\"000e489c7e46345eabd8486f748a69e64f3cdc956ae6b5d2faa82b935c9b5a6b\",\"nonce\":710,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-03-17 13:52:18.562\",\"tx\":\"Bill pays Clara 50 DSCoin\",\"previousHash\":\"006ea898348865e93a78b10136458b173f0dcde48a3388150c2ecd3db8abee09\",\"nonce\":92,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-03-17 13:52:25.02\",\"tx\":\"Clara pays Daisy 10 DS Coin\",\"previousHash\":\"004c4eec5fbac1c706417cfa0dd311425ea1e05c2e92275ac6599ae3d523aaf6\",\"nonce\":519,\"difficulty\":2}]"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

4

corrupt the Blockchain

Enter block ID of block to corrupt

1

Enter new data for block 1

Alice pays Bill 76 DSCoin

"Block 1 now holds Alice pays Bill 76 DSCoin"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

3

"[{\"index\":0,\"timestamp\":\"2023-03-17 13:51:48.715\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":212,\"difficulty\":2},{\"index\":1,\"timestamp\":\"2023-03-17 13:52:10.226\",\"tx\":\"Alice pays Bill 76 DSCoin\",\"previousHash\":\"000e489c7e46345eabd8486f748a69e64f3cdc956ae6b5d2faa82b935c9b5a6b\",\"nonce\":710,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-03-17 13:52:18.562\",\"tx\":\"Bill pays Clara 50 DSCoin\",\"previousHash\":\"006ea898348865e93a78b10136458b173f0dcde48a3388150c2ecd3db8abee09\",\"nonce\":92,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-03-17 13:52:25.02\",\"tx\":\"Clara pays Daisy 10 DS Coin\",\"previousHash\":\"004c4eec5fbac1c706417cfa0dd311425ea1e05c2e92275ac6599ae3d523aaf6\",\"nonce\":519,\"difficulty\":2}]"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

2

"Total execution time to verify the chain was 3 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

5

"Total execution time required to repair the chain was 5 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

2

"Total execution time to verify the chain was 0 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

1

Enter difficulty > 0

4

Enter transaction:

Daisy pays Sean 25 DSCoin

"Total execution time to add this block was 195 milliseconds"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

0

Current size of chain: 5

Difficulty of most recent block: 4

Total difficulty for all blocks: 12

Approximate hashes per second on this machine : 2000000

Expected total hashes required for the whole chain: 66560.0

Nonce for most recent block: 88412

Chain hash: "000028b90355b5f5acf105695b68a34676649aadb4c5d11e270e1ce036d1c522"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

6

Process finished with exit code 0

***Task 1 Server Side Execution***

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea\_rt.jar=61146:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ruthpeter/ds/Project3/Project3Task1/target/classes:/Users/ruthpeter/.m2/repository/com/google/code/gson/gson/2.2.4/gson-2.2.4.jar:/Users/ruthpeter/.m2/repository/org/json/json/20230227/json-20230227.jar blockchaintask1.EchoServerTCP

Blockchain server running

We have a visitor

Response : {"selection":0,"size":1,"chainHash":"000e489c7e46345eabd8486f748a69e64f3cdc956ae6b5d2faa82b935c9b5a6b","totalHashes":256.0,"totalDiff":2,"recentNonce":212,"diff":2,"hps":2000000}

Adding a block

Setting response to : {"selection":1,"response":"Total execution time to add this block was 7 milliseconds"}

Adding a block

Setting response to : {"selection":1,"response":"Total execution time to add this block was 3 milliseconds"}

Adding a block

Setting response to : {"selection":1,"response":"Total execution time to add this block was 6 milliseconds"}

Verifying entire chain

Chain verification: TRUE

Setting response to: {"selection":2,"response":"Total execution time to verify the chain was 1 milliseconds"}

View the Blockchain

Setting response to: {"selection":3,"response":"[{\"index\":0,\"timestamp\":\"2023-03-17 13:51:48.715\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":212,\"difficulty\":2},{\"index\":1,\"timestamp\":\"2023-03-17 13:52:10.226\",\"tx\":\"Alice pays Bill 100 DSCoin\",\"previousHash\":\"000e489c7e46345eabd8486f748a69e64f3cdc956ae6b5d2faa82b935c9b5a6b\",\"nonce\":710,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-03-17 13:52:18.562\",\"tx\":\"Bill pays Clara 50 DSCoin\",\"previousHash\":\"006ea898348865e93a78b10136458b173f0dcde48a3388150c2ecd3db8abee09\",\"nonce\":92,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-03-17 13:52:25.02\",\"tx\":\"Clara pays Daisy 10 DS Coin\",\"previousHash\":\"004c4eec5fbac1c706417cfa0dd311425ea1e05c2e92275ac6599ae3d523aaf6\",\"nonce\":519,\"difficulty\":2}]"}

Corrupt the Blockchain

Setting response to: {"selection":4,"response":"Block 1 now holds Alice pays Bill 76 DSCoin"}

View the Blockchain

Setting response to: {"selection":3,"response":"[{\"index\":0,\"timestamp\":\"2023-03-17 13:51:48.715\",\"tx\":\"Genesis\",\"previousHash\":\"\",\"nonce\":212,\"difficulty\":2},{\"index\":1,\"timestamp\":\"2023-03-17 13:52:10.226\",\"tx\":\"Alice pays Bill 76 DSCoin\",\"previousHash\":\"000e489c7e46345eabd8486f748a69e64f3cdc956ae6b5d2faa82b935c9b5a6b\",\"nonce\":710,\"difficulty\":2},{\"index\":2,\"timestamp\":\"2023-03-17 13:52:18.562\",\"tx\":\"Bill pays Clara 50 DSCoin\",\"previousHash\":\"006ea898348865e93a78b10136458b173f0dcde48a3388150c2ecd3db8abee09\",\"nonce\":92,\"difficulty\":2},{\"index\":3,\"timestamp\":\"2023-03-17 13:52:25.02\",\"tx\":\"Clara pays Daisy 10 DS Coin\",\"previousHash\":\"004c4eec5fbac1c706417cfa0dd311425ea1e05c2e92275ac6599ae3d523aaf6\",\"nonce\":519,\"difficulty\":2}]"}

Verifying entire chain

Chain verification: FALSE

Improper hash on node 1 Does not begin with 00

Setting response to: {"selection":2,"response":"Total execution time to verify the chain was 3 milliseconds"}

Repairing the entire chain

Setting response to: {"selection":5,"response":"Total execution time required to repair the chain was 5 milliseconds"}

Verifying entire chain

Chain verification: TRUE

Setting response to: {"selection":2,"response":"Total execution time to verify the chain was 0 milliseconds"}

Adding a block

Setting response to : {"selection":1,"response":"Total execution time to add this block was 195 milliseconds"}

Response : {"selection":0,"size":5,"chainHash":"000028b90355b5f5acf105695b68a34676649aadb4c5d11e270e1ce036d1c522","totalHashes":66560.0,"totalDiff":12,"recentNonce":88412,"diff":4,"hps":2000000}

Process finished with exit code 0

***Task 1 Client Source Code***

EchoClientTCP.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
//EchoServerTCP.java from Coulouris text  
  
import com.google.gson.JsonObject;  
import com.google.gson.JsonParser;  
  
import java.io.\*;  
import java.net.Socket;  
import java.util.Scanner;  
  
public class EchoClientTCP {  
  
 static Socket *clientSocket* = null;  
  
 public static void main(String args[]) {  
 // arguments supply hostname  
 Scanner sc = new Scanner(System.*in*);  
 try {  
 System.*out*.println("The TCP client is running.");  
  
 *clientSocket* = new Socket("localhost", 7777);  
  
 //to read from the server  
 BufferedReader in = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
  
 //to write to the server  
 PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream())));  
  
 int option = 0;  
  
 while (true) {  
  
 System.*out*.println("0. View basic blockchain status.\n" +  
 "1. Add a transaction to the blockchain.\n" +  
 "2. Verify the blockchain.\n" +  
 "3. View the blockchain.\n" +  
 "4. Corrupt the chain.\n" +  
 "5. Hide the corruption by repairing the chain.\n" +  
 "6. Exit");  
  
 option = sc.nextInt();  
  
 switch (option) {  
 //blockchain status  
 case 0: {  
 RequestMessage requestMessage = new RequestMessage(option);  
  
 out.println(requestMessage.getRequestJson());  
 out.flush();  
  
 String reply = in.readLine();  
  
 //Parsing response object from server as json object  
 JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();  
  
 System.*out*.println("Current size of chain: " + jsonReply.get("size"));  
 System.*out*.println("Difficulty of most recent block: " + jsonReply.get("diff"));  
 System.*out*.println("Total difficulty for all blocks: " + jsonReply.get("totalDiff"));  
 System.*out*.println("Approximate hashes per second on this machine : " + jsonReply.get("hps"));  
 System.*out*.println("Expected total hashes required for the whole chain: " + jsonReply.get("totalHashes"));  
 System.*out*.println("Nonce for most recent block: " + jsonReply.get("recentNonce"));  
 System.*out*.println("Chain hash: " + jsonReply.get("chainHash") + "\n");  
  
 break;  
 }  
  
 //add block to blockchain  
 case 1: {  
 System.*out*.println("Enter difficulty > 0 \n");  
  
 int difficulty = sc.nextInt();  
 sc.nextLine();  
 System.*out*.println("Enter transaction: \n");  
 String transaction = sc.nextLine();  
  
 RequestMessage requestMessage = new RequestMessage(option, difficulty, transaction);  
  
 out.println(requestMessage.getRequestJson());  
 out.flush();  
  
 String reply = in.readLine();  
 //Parsing response object from server as json object  
 JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();  
 System.*out*.println(jsonReply.get("response"));  
  
 break;  
  
 }  
  
  
 // verify, view, and repair blockchain  
 case 2, 3, 5: {  
  
 RequestMessage requestMessage = new RequestMessage(option);  
  
 out.println(requestMessage.getRequestJson());  
 out.flush();  
  
 String reply = in.readLine();  
 //Parsing response object from server as json object  
 JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();  
 System.*out*.println(jsonReply.get("response"));  
  
 break;  
 }  
  
  
 //corrupt blockchain at index  
 case 4: {  
 System.*out*.println("corrupt the Blockchain");  
 System.*out*.println("Enter block ID of block to corrupt");  
 int index = sc.nextInt();  
 sc.nextLine();  
 System.*out*.println("Enter new data for block " + index);  
 String transaction = sc.nextLine();  
  
 RequestMessage requestMessage = new RequestMessage(option, index, transaction);  
 out.println(requestMessage.getRequestJson());  
 out.flush();  
  
 String reply = in.readLine();  
 //Parsing response object from server as json object  
 JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();  
 System.*out*.println(jsonReply.get("response"));  
 break;  
 }  
  
  
 //exit program  
 case 6:  
 RequestMessage requestMessage = new RequestMessage(option);  
 out.println(requestMessage.getRequestJson());  
 out.flush();  
 System.*exit*(0);  
  
  
 }  
 }  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
 } finally {  
 try {  
 if (*clientSocket* != null) {  
 *clientSocket*.close();  
 }  
 } catch (IOException e) {  
 // ignore exception on close  
 }  
 }  
 }  
}

RequestMessage.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
import com.google.gson.JsonObject;  
  
public class RequestMessage {  
  
 private JsonObject requestJson = new JsonObject();  
  
  
 // options 0, 2, 3, 4, 6  
 */\*\*  
 \* Constructor for single operation  
 \*  
 \** ***@param*** *op  
 \*/* public RequestMessage(int op) {  
 requestJson.addProperty("op", op);  
 }  
  
 // options 1 , 4  
 */\*\*  
 \* Constructor to send additional property and transaction details  
 \*  
 \** ***@param*** *op  
 \** ***@param*** *property  
 \** ***@param*** *transaction  
 \*/* public RequestMessage(int op, int property, String transaction) {  
  
 requestJson.addProperty("op", op);  
 if (op == 1)  
 requestJson.addProperty("difficulty", property);  
  
 if (op == 4)  
 requestJson.addProperty("index", property);  
  
 requestJson.addProperty("transaction", transaction);  
  
 }  
  
 */\*\*  
 \* Getter method for requestJson  
 \*  
 \** ***@return*** *\*/* public JsonObject getRequestJson() {  
 return requestJson;  
 }  
}

ResponseMessage.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
import com.google.gson.JsonObject;  
  
import java.math.BigInteger;  
  
public class ResponseMessage {  
  
 private JsonObject responseJson = new JsonObject();  
  
  
 // option 0  
  
 */\*\*  
 \* Constructor for ResponseMessage option 0 with all parameters  
 \*  
 \** ***@param*** *option  
 \** ***@param*** *chainSize  
 \** ***@param*** *difficulty  
 \** ***@param*** *totalDifficulty  
 \** ***@param*** *hashesPerSecond  
 \** ***@param*** *totalExpectedHashes  
 \** ***@param*** *nonce  
 \** ***@param*** *chainHash  
 \*/* public ResponseMessage(int option, int chainSize, int difficulty, int totalDifficulty, int hashesPerSecond, double totalExpectedHashes, BigInteger nonce, String chainHash) {  
 responseJson.addProperty("selection", option);  
 responseJson.addProperty("size", chainSize);  
 responseJson.addProperty("chainHash", chainHash);  
 responseJson.addProperty("totalHashes", totalExpectedHashes);  
 responseJson.addProperty("totalDiff", totalDifficulty);  
 responseJson.addProperty("recentNonce", nonce);  
 responseJson.addProperty("diff", difficulty);  
 responseJson.addProperty("hps", hashesPerSecond);  
 }  
  
  
 // option 1  
  
 */\*\*  
 \* Constructor for ResponseMessage with selection message and response  
 \*  
 \** ***@param*** *selection  
 \** ***@param*** *response  
 \*/* public ResponseMessage(int selection , String response) {  
 responseJson.addProperty("selection", selection);  
 responseJson.addProperty("response", response);  
 }  
  
  
 */\*\*  
 \* getter method for ResponseJson  
 \*  
 \** ***@return*** *\*/* public JsonObject getResponseJson() {  
 return responseJson;  
 }  
}

***Task 1 Server Source Code***

EchoServerTcp.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
  
//EchoServerTCP.java from Coulouris text  
  
import com.google.gson.JsonObject;  
import com.google.gson.JsonParser;  
  
import java.io.BufferedWriter;  
import java.io.IOException;  
import java.io.OutputStreamWriter;  
import java.io.PrintWriter;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.security.NoSuchAlgorithmException;  
import java.sql.Timestamp;  
import java.util.Scanner;  
  
public class EchoServerTCP {  
  
 public static void main(String args[]) {  
  
 Socket clientSocket = null;  
  
 System.*out*.println("Blockchain server running");  
 try {  
 Scanner sc = new Scanner(System.*in*);  
 int serverPort = 7777; // Read user input  
 ServerSocket listenSocket = new ServerSocket(serverPort);  
  
 /\*  
 \* Block waiting for a new connection request from a client.  
 \* When the request is received, "accept" it, and the rest  
 \* the tcp protocol handshake will then take place, making  
 \* the socket ready for reading and writing.  
 \*/  
 clientSocket = listenSocket.accept();  
 // If we get here, then we are now connected to a client.  
  
 // Set up "in" to read from the client socket  
 Scanner in;  
 in = new Scanner(clientSocket.getInputStream());  
  
 // Set up "out" to write to the client socket  
 PrintWriter out;  
 out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));  
  
 /\*  
 \* Forever,  
 \* read a line from the socket  
 \* print it to the console  
 \* echo it (i.e. write it) back to the client  
 \*/  
  
 BlockChain chain = new BlockChain();  
 int option = 0;  
  
  
 Block genesis = new Block(0, chain.getTime(), "Genesis", 2);  
 genesis.setPreviousHash("");  
 genesis.proofOfWork();  
  
 chain.computeHashesPerSecond();  
 chain.addBlock(genesis);  
  
 System.*out*.println("We have a visitor");  
  
 while (true) {  
 JsonObject clientReply = new JsonParser().parse(in.nextLine()).getAsJsonObject();  
 option = clientReply.get("op").getAsInt();  
  
 switch (option) {  
 case 0: {  
 //Status  
 ResponseMessage responseMessage = new ResponseMessage(option, chain.getChainSize(), chain.getLatestBlock().getDifficulty(), chain.getTotalDifficulty(), chain.getHashesPerSecond(), chain.getTotalExpectedHashes(), chain.getLatestBlock().getNonce(), chain.getChainHash());  
 System.*out*.println("Response : " + responseMessage.getResponseJson());  
 out.println(responseMessage.getResponseJson());  
 out.flush();  
  
 break;  
 }  
  
  
 case 1: {  
  
 System.*out*.println("Adding a block");  
 int difficulty = clientReply.get("difficulty").getAsInt();  
 String transaction = clientReply.get("transaction").getAsString();  
  
 Timestamp start = chain.getTime();  
 Block newBlock = new Block(chain.getChainSize(), chain.getTime(), transaction, difficulty);  
 newBlock.setPreviousHash(chain.getChainHash());  
 newBlock.proofOfWork();  
 chain.addBlock(newBlock);  
  
 Timestamp end = chain.getTime();  
  
 ResponseMessage responseMessage = new ResponseMessage(option, "Total execution time to add this block was " + (end.getTime() - start.getTime()) + " milliseconds");  
 System.*out*.println("Setting response to : " + responseMessage.getResponseJson());  
 out.println(responseMessage.getResponseJson());  
 out.flush();  
  
 break;  
 }  
  
 case 2: {  
 System.*out*.println("Verifying entire chain");  
 Timestamp start = chain.getTime();  
 System.*out*.println("Chain verification: " + chain.isChainValid());  
 Timestamp end = chain.getTime();  
 ResponseMessage responseMessage = new ResponseMessage(option, "Total execution time to verify the chain was " + (end.getTime() - start.getTime()) + " milliseconds");  
 System.*out*.println("Setting response to: " + responseMessage.getResponseJson());  
 out.println(responseMessage.getResponseJson());  
 out.flush();  
 break;  
 }  
  
 case 3: {  
 System.*out*.println("View the Blockchain");  
 ResponseMessage responseMessage = new ResponseMessage(option, chain.toString());  
 System.*out*.println("Setting response to: " + responseMessage.getResponseJson());  
 out.println(responseMessage.getResponseJson());  
 out.flush();  
 break;  
  
 }  
  
 case 4: {  
 System.*out*.println("Corrupt the Blockchain");  
 int index = clientReply.get("index").getAsInt();  
 String transaction = clientReply.get("transaction").getAsString();  
  
 chain.getBlock(index).setData(transaction);  
  
 ResponseMessage responseMessage = new ResponseMessage(option, "Block " + index + " now holds " + transaction);  
 System.*out*.println("Setting response to: " + responseMessage.getResponseJson());  
 out.println(responseMessage.getResponseJson());  
 out.flush();  
 break;  
 }  
  
 case 5: {  
 System.*out*.println("Repairing the entire chain");  
 Timestamp start = chain.getTime();  
 chain.repairChain();  
 Timestamp end = chain.getTime();  
  
 ResponseMessage responseMessage = new ResponseMessage(option, "Total execution time required to repair the chain was " + (end.getTime() - start.getTime()) + " milliseconds");  
 System.*out*.println("Setting response to: " + responseMessage.getResponseJson());  
 out.println(responseMessage.getResponseJson());  
 out.flush();  
 break;  
 }  
 case 6:  
 System.*exit*(0);  
 }  
 }  
  
 // Handle exceptions  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
  
 // If quitting (typically by you sending quit signal) clean up sockets  
 } catch (NoSuchAlgorithmException e) {  
 throw new RuntimeException(e);  
 } finally {  
 try {  
 if (clientSocket != null) {  
 clientSocket.close();  
 }  
 } catch (IOException e) {  
 // ignore exception on close  
 }  
 }  
 }  
}

RequestMessage.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
import com.google.gson.JsonObject;  
  
public class RequestMessage {  
  
 private JsonObject requestJson = new JsonObject();  
  
  
 // options 0, 2, 3, 4, 6  
 */\*\*  
 \* Constructor for single operation  
 \*  
 \** ***@param*** *op  
 \*/* public RequestMessage(int op) {  
 requestJson.addProperty("op", op);  
 }  
  
 // options 1 , 4  
 */\*\*  
 \* Constructor to send additional property and transaction details  
 \*  
 \** ***@param*** *op  
 \** ***@param*** *property  
 \** ***@param*** *transaction  
 \*/* public RequestMessage(int op, int property, String transaction) {  
  
 requestJson.addProperty("op", op);  
 if (op == 1)  
 requestJson.addProperty("difficulty", property);  
  
 if (op == 4)  
 requestJson.addProperty("index", property);  
  
 requestJson.addProperty("transaction", transaction);  
  
 }  
  
 */\*\*  
 \* Getter method for requestJson  
 \*  
 \** ***@return*** *\*/* public JsonObject getRequestJson() {  
 return requestJson;  
 }  
}

ResponseMessage.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
import com.google.gson.JsonObject;  
  
import java.math.BigInteger;  
  
public class ResponseMessage {  
  
 private JsonObject responseJson = new JsonObject();  
  
  
 // option 0  
  
 */\*\*  
 \* Constructor for ResponseMessage option 0 with all parameters  
 \*  
 \** ***@param*** *option  
 \** ***@param*** *chainSize  
 \** ***@param*** *difficulty  
 \** ***@param*** *totalDifficulty  
 \** ***@param*** *hashesPerSecond  
 \** ***@param*** *totalExpectedHashes  
 \** ***@param*** *nonce  
 \** ***@param*** *chainHash  
 \*/* public ResponseMessage(int option, int chainSize, int difficulty, int totalDifficulty, int hashesPerSecond, double totalExpectedHashes, BigInteger nonce, String chainHash) {  
 responseJson.addProperty("selection", option);  
 responseJson.addProperty("size", chainSize);  
 responseJson.addProperty("chainHash", chainHash);  
 responseJson.addProperty("totalHashes", totalExpectedHashes);  
 responseJson.addProperty("totalDiff", totalDifficulty);  
 responseJson.addProperty("recentNonce", nonce);  
 responseJson.addProperty("diff", difficulty);  
 responseJson.addProperty("hps", hashesPerSecond);  
 }  
  
  
 // option 1  
  
 */\*\*  
 \* Constructor for ResponseMessage with selection message and response  
 \*  
 \** ***@param*** *selection  
 \** ***@param*** *response  
 \*/* public ResponseMessage(int selection , String response) {  
 responseJson.addProperty("selection", selection);  
 responseJson.addProperty("response", response);  
 }  
  
  
 */\*\*  
 \* getter method for ResponseJson  
 \*  
 \** ***@return*** *\*/* public JsonObject getResponseJson() {  
 return responseJson;  
 }  
}

Block.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*/*package blockchaintask1;  
  
import com.google.gson.JsonObject;  
  
import java.math.BigInteger;  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.sql.Timestamp;  
  
public class Block {  
  
 // the position of the block on the chain  
 private int index;  
  
 //time of the block's creation  
 private Timestamp timestamp;  
  
 //block's single transaction details  
 private String data;  
  
 //the SHA256 hash of a block's parent  
 private String previousHash;  
  
 //determined by POW routine  
 private BigInteger nonce;  
  
 //minimum number of left most hex digits needed by a proper hash  
 private int difficulty;  
  
 public Block(int index, Timestamp timestamp, String data, int difficulty) {  
 this.index = index;  
 this.timestamp = timestamp;  
 this.data = data;  
 this.difficulty = difficulty;  
 this.nonce = BigInteger.*ZERO*;  
 }  
  
  
 */\*\*  
 \* This method computes a hash of the concatenation of the index,  
 \* timestamp, data, previousHash, nonce, and difficulty.  
 \*  
 \** ***@return*** *a String holding Hexadecimal characters  
 \*/* public String calculateHash() throws NoSuchAlgorithmException {  
 String parentString = String.*valueOf*(this.index) + this.timestamp + this.data + this.previousHash + this.nonce + this.difficulty;  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 byte[] encodedHash = md.digest(  
 parentString.getBytes(StandardCharsets.*UTF\_8*));  
 return BlockHelper.*bytesToHex*(encodedHash);  
 }  
  
  
 */\*\*  
 \* This method returns the nonce for this block  
 \*  
 \** ***@return*** *a BigInteger representing the nonce for this block  
 \*/* public BigInteger getNonce() {  
 return nonce;  
 }  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *difficulty  
 \*/* public int getDifficulty() {  
 return difficulty;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *difficulty - determines how much work is required to produce a proper hash  
 \*/* public void setDifficulty(int difficulty) {  
 this.difficulty = difficulty;  
 }  
  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *index  
 \*/* public int getIndex() {  
 return index;  
 }  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *timestamp of this block  
 \*/* public Timestamp getTimestamp() {  
 return timestamp;  
 }  
  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *this block's transaction  
 \*/* public String getData() {  
 return data;  
 }  
  
 */\*\*  
 \* Simple getter method  
 \*  
 \** ***@return*** *previous hash  
 \*/* public String getPreviousHash() {  
 return previousHash;  
 }  
  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *index - the index of this block in the chain  
 \*/* public void setIndex(int index) {  
 this.index = index;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *timestamp - of when this block was created  
 \*/* public void setTimestamp(Timestamp timestamp) {  
 this.timestamp = timestamp;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *data - represents the transaction held by this block  
 \*/* public void setData(String data) {  
 this.data = data;  
 }  
  
 */\*\*  
 \* Simple setter method  
 \*  
 \** ***@param*** *previousHash - a hashpointer to this block's parent  
 \*/* public void setPreviousHash(String previousHash) {  
 this.previousHash = previousHash;  
 }  
  
  
 */\*\*  
 \** ***@return*** *A JSON representation of all of this block's data is returned  
 \*/* @Override  
 public String toString() {  
  
 JsonObject jsonObject = new JsonObject();  
  
 jsonObject.addProperty("index", index);  
 jsonObject.addProperty("timestamp", String.*valueOf*(timestamp));  
 jsonObject.addProperty("tx", data);  
 jsonObject.addProperty("previousHash", previousHash);  
 jsonObject.addProperty("nonce", nonce);  
 jsonObject.addProperty("difficulty", difficulty);  
 return jsonObject.toString();  
 }  
  
 */\*\*  
 \* The proof of work methods finds a good hash.  
 \*  
 \** ***@return*** *a String with a hash that has the appropriate number of leading hex zeroes.  
 \*/* public String proofOfWork() throws NoSuchAlgorithmException {  
  
 String hexHash = calculateHash();  
  
 String matchString = "";  
 //this.difficulty  
  
 for (int i = 0; i < this.difficulty; i++) {  
 matchString = matchString + "0";  
 }  
  
 while(!hexHash.substring(0, this.difficulty).equalsIgnoreCase(matchString)){  
 this.nonce = this.nonce.add(BigInteger.*ONE*);  
 hexHash = calculateHash();  
 }  
  
 return hexHash;  
  
 }  
  
}

BlockChain.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*/*package blockchaintask1;  
  
import com.google.gson.Gson;  
import com.google.gson.JsonArray;  
import com.google.gson.JsonElement;  
  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.sql.Timestamp;  
import java.util.ArrayList;  
import java.util.List;  
  
public class BlockChain {  
  
 private List<Block> blockList;  
  
 private String chainHash;  
  
 private int hashesPerSecond;  
  
 Gson gson = new Gson();  
  
 public BlockChain() {  
 this.blockList = new ArrayList<>();  
 this.chainHash = "";  
 this.hashesPerSecond = 0;  
  
 }  
  
 */\*\*  
 \** ***@return*** *the chain hash  
 \*/* public String getChainHash() {  
 return chainHash;  
 }  
  
 */\*\*  
 \** ***@return*** *the current system time  
 \*/* public Timestamp getTime() {  
 return new Timestamp(System.*currentTimeMillis*());  
 }  
  
 */\*\*  
 \* a reference to the most recently added Block  
 \*  
 \** ***@return*** *\*/* public Block getLatestBlock() {  
 return blockList.get(blockList.size() - 1);  
 }  
  
 public int getChainSize() {  
 return blockList.size();  
 }  
  
 */\*\*  
 \* This method computes exactly 2 million hashes and times how long that process takes  
 \*/* public void computeHashesPerSecond() throws NoSuchAlgorithmException {  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 int i = 0;  
 Timestamp startTime = getTime();  
 while (i < 2000000) {  
 byte[] encodedHash = md.digest(  
 "00000000".getBytes(StandardCharsets.*UTF\_8*));  
 i += 1;  
 }  
 Timestamp endTime = getTime();  
 this.hashesPerSecond = 2000000 / (endTime.compareTo(startTime));  
  
 }  
  
 */\*\*  
 \* get hashes per second  
 \*  
 \** ***@return*** *the instance variable approximating the number of hashes per second  
 \*/* public int getHashesPerSecond() {  
 return this.hashesPerSecond;  
 }  
  
 */\*\*  
 \* Method to add new block to blockChain  
 \* block's previous hash must hold the hash of the most recently added block  
 \* the new block becomes the most recently added block on the BlockChain  
 \* The SHA256 hash of every block must exhibit proof of work  
 \*  
 \** ***@param*** *newBlock  
 \** ***@throws*** *NoSuchAlgorithmException  
 \*/* public void addBlock(Block newBlock) throws NoSuchAlgorithmException {  
 if (this.blockList.size() > 0)  
 newBlock.setPreviousHash(getChainHash());  
 else  
 newBlock.setPreviousHash("");  
 blockList.add(newBlock);  
 this.chainHash = newBlock.proofOfWork();  
 }  
  
 */\*\*  
 \* This method uses the toString method defined on each individual block  
 \*  
 \** ***@return*** *a String representation of the entire chain is returned  
 \*/* @Override  
 public String toString() {  
  
 JsonArray jsonArray = new JsonArray();  
  
 for (Block b : blockList) {  
 jsonArray.add(gson.fromJson(b.toString(), JsonElement.class));  
 }  
  
 return jsonArray.toString();  
  
 }  
  
 */\*\*  
 \* return block at position i  
 \*  
 \** ***@param*** *i  
 \** ***@return*** *\*/* public Block getBlock(int i) {  
 return blockList.get(i);  
 }  
  
  
 */\*\*  
 \* Compute and return the total difficulty of all blocks on the chain. Each block knows its own difficulty  
 \*  
 \** ***@return*** *totalDifficulty  
 \*/* public int getTotalDifficulty() {  
 int totalDifficulty = 0;  
 for (Block block : blockList) {  
 totalDifficulty += block.getDifficulty();  
 }  
 return totalDifficulty;  
 }  
  
 */\*\*  
 \* Compute and return the expected number of hashes required for the entire chain.  
 \** ***@return*** *totalExpectedHashes  
 \*/* public double getTotalExpectedHashes() {  
 double totalExpectedHashes = 0;  
 for (Block block : blockList)  
 totalExpectedHashes += Math.*pow*(16, block.getDifficulty()); // 16 (16 hex characters) ^ difficulty of block  
 return totalExpectedHashes;  
 }  
  
 */\*\*  
 \* checks if the hash has requisite number of leftmost zeroes as specified in difficulty for the entire chain  
 \*  
 \** ***@return*** *"TRUE" if the chain is valid, otherwise return a string with an appropriate error message  
 \** ***@throws*** *NoSuchAlgorithmException  
 \*/* public String isChainValid() throws NoSuchAlgorithmException {  
 //chain contains only 1 block , i.e. genesis  
 if (blockList.size() == 1) {  
 Block genesisBlock = this.blockList.get(0);  
 String hash = genesisBlock.calculateHash();  
 //calculate prefix based on difficulty, number of leading zeroes based on the difficulty value  
 String prefix = new String(new char[genesisBlock.getDifficulty()]).replace("\0", "0");  
 if (!hash.substring(0, genesisBlock.getDifficulty()).equals(prefix)) {  
 return "FALSE \n Improper hash on genesis node";  
 } else if (!chainHash.equals(hash)) {  
 return "FALSE \n Chain hash and computed hash do not match";  
 } else {  
 return "TRUE";  
 }  
 }  
  
 //more than 1 block  
 if (blockList.size() > 1) {  
 for (int i = 1; i < blockList.size(); i++) {  
  
 Block currentBlock = this.blockList.get(i);  
 Block previousBlock = this.blockList.get(i - 1);  
  
  
 String hash = currentBlock.calculateHash();  
 String hashPointer = currentBlock.getPreviousHash();  
 //calculate prefix based on difficulty, number of leading zeroes based on the difficulty value  
 String prefix = new String(new char[currentBlock.getDifficulty()]).replace("\0", "0");  
  
 if (!hash.substring(0, currentBlock.getDifficulty()).equals(prefix))  
 return "FALSE \n Improper hash on node " + i + " Does not begin with " + prefix;  
 //check proof of work / leading zeros  
 else if (!hashPointer.equals(previousBlock.calculateHash()))  
 return "FALSE \n Improper previous hash";  
 }  
 }  
  
 //chain hash , check the last element added to to the blocklist  
 if (!chainHash.equals(blockList.get(blockList.size() - 1).calculateHash())) {  
 return "Chain hash error";  
 }  
  
 return "TRUE";  
 }  
  
 */\*\*  
 \* This routine repairs the chain. It checks the hashes of each block and ensures that any illegal hashes are recomputed.  
 \* After this routine is run, the chain will be valid. The routine does not modify any difficulty values.  
 \* It computes new proof of work based on the difficulty specified in the Block.  
 \*  
 \** ***@throws*** *NoSuchAlgorithmException  
 \*/* public void repairChain() throws NoSuchAlgorithmException {  
  
 //genesis block  
 if (blockList.size() == 1) {  
 //Reset previous hash and recompute proof of work  
 blockList.get(0).setPreviousHash("");  
 blockList.get(0).proofOfWork();  
 }  
  
  
 if (blockList.size() > 1) {  
 for (int i = 1; i < blockList.size(); i++) {  
 //reset previous hash and recompute proof of work  
 blockList.get(i).setPreviousHash(blockList.get(i - 1).calculateHash());  
 blockList.get(i).proofOfWork();  
 }  
  
 //reset chain hash  
 this.chainHash = blockList.get(blockList.size() - 1).calculateHash();  
 }  
 }  
  
}

BlockHelper.java

*/\*\*  
 \* Name : Ruth Peter  
 \* Andrew id : rpeter  
 \*  
 \*/*package blockchaintask1;  
  
public class BlockHelper {  
  
  
 //https://www.baeldung.com/sha-256-hashing-java  
 public static String bytesToHex(byte[] hash) {  
 StringBuilder hexString = new StringBuilder(2 \* hash.length);  
 for (int i = 0; i < hash.length; i++) {  
 String hex = Integer.*toHexString*(0xff & hash[i]);  
 if(hex.length() == 1) {  
 hexString.append('0');  
 }  
 hexString.append(hex);  
 }  
 return hexString.toString();  
 }  
}

***Project3Task2***

**Transaction details of funds from dispenser**

GET https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/T5TMCVYATZDHQKLR4SYFFXMRCXEG22Z7SQWXWLSDHV5AUVNTFVUA

HTTP/1.1 200 OK

server: nginx

date: Fri, 17 Mar 2023 19:02:36 GMT

content-type: application/json; charset=UTF-8

content-length: 728

vary: Origin

access-control-allow-methods: GET,POST,OPTIONS

access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-Stats, Authorization

cache-control: no-store, no-cache, must-revalidate, private

{

"current-round": 28477128,

"transaction": {

"close-rewards": 0,

"closing-amount": 0,

"confirmed-round": 28476134,

"fee": 1000,

"first-valid": 28476132,

"genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",

"genesis-id": "testnet-v1.0",

"id": "T5TMCVYATZDHQKLR4SYFFXMRCXEG22Z7SQWXWLSDHV5AUVNTFVUA",

"intra-round-offset": 0,

"last-valid": 28477132,

"payment-transaction": {

"amount": 10000000,

"close-amount": 0,

"receiver": "24ST427BEAUYGOXOUWB3IL5V47YLENYQYQSQPCH3KIVLNLLURTE7KFUH7M"

},

"receiver-rewards": 0,

"round-time": 1679076148,

"sender": "DISPE57MNLYKOMOK3H5IMBAYOYW3YL2CSI6MDOG3RDXSMET35DG4W6SOTI",

"sender-rewards": 0,

"signature": {

"sig": "iAInLrhzkUK05tZQlyLrADPQ1ccdCKdGltqKS6kalfmmVw3111vsXT30tNnHqgz7ooV68asvk+M7/HjNvGhFDw=="

},

"tx-type": "pay"

}

}

Response file saved.

> 2023-03-17T150236.200.json

Response code: 200 (OK); Time: 385ms (385 ms); Content length: 728 bytes (728 B)

**Transaction details for sending 5 Algos**

GET https://algoindexer.testnet.algoexplorerapi.io/v2/transactions/O4JFRTTRGXKRS5YCL7NLWBPYWFVDDN5ADGLBPD7QXT6XRH7NZY2A

HTTP/1.1 200 OK

server: nginx

date: Fri, 17 Mar 2023 19:04:38 GMT

content-type: application/json; charset=UTF-8

content-length: 761

vary: Origin

access-control-allow-methods: GET,POST,OPTIONS

access-control-allow-headers: Content-Type, X-Disable-Tracking, X-Algoexplorer-Api-Key, X-Debug-Stats, Authorization

cache-control: no-store, no-cache, must-revalidate, private

{

"current-round": 28477162,

"transaction": {

"close-rewards": 0,

"closing-amount": 0,

"confirmed-round": 28476414,

"fee": 1000,

"first-valid": 28476412,

"genesis-hash": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",

"genesis-id": "testnet-v1.0",

"id": "O4JFRTTRGXKRS5YCL7NLWBPYWFVDDN5ADGLBPD7QXT6XRH7NZY2A",

"intra-round-offset": 3,

"last-valid": 28477412,

"note": "QW5kcmV3IGlkIDogcnBldGVy",

"payment-transaction": {

"amount": 5000000,

"close-amount": 0,

"receiver": "K2EP3LIPR3KEI7QOVW3UHLN6JGASMF442YRI5IPO6N6UWPUVNZJ6BVFT4U"

},

"receiver-rewards": 0,

"round-time": 1679077157,

"sender": "24ST427BEAUYGOXOUWB3IL5V47YLENYQYQSQPCH3KIVLNLLURTE7KFUH7M",

"sender-rewards": 0,

"signature": {

"sig": "rNndKsIEKZjmJZwAcdl9VW7kwG0x4knyD0ELbJ3Ch+pa2+eERJ+svqM2CrfiDSIbip7ByyXxCd9r5IhR5oslCg=="

},

"tx-type": "pay"

}

}

Response file saved.

> 2023-03-17T150438.200.json

Response code: 200 (OK); Time: 421ms (421 ms); Content length: 761 bytes (761 B)