

## Project 3

Name: Ariane Correa  
Andrew ID: ajcorrea

### Task 0

#### **Console Output:**

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -  
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52522:/Applications/IntelliJ  
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath  
/Users/arianecorrea/IdeaProjects/Project3Task0/target/classes:/Users/arianecorrea/.m2/repos  
itory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.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: 3205

Expected total hashes required for the whole chain: 256.0

Nonce for the most recent block: 216

Chain hash: 0073cf07aeacb687592854d2b6b6015c15f4dd8c7d64771c52e910413e59e0da

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:

Mike pays Marty 100 DS<sup>C</sup>oin

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:

Marty pays Joe 50 DS<sup>C</sup>oin

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:

Joe pays Andy 10 DS Coin

Total execution time to add this block was 10 milliseconds

- 0. View basic blockchain status.
- 1. Add a transaction to the blockchain.
- 2. Verify the blockchain.
- 3. View the blockchain.
- 4. Corrupt the chain.
- 5. Hide the corruption by repairing the chain.
- 6. Exit

2

Chain verification: TRUE

Total execution time to verify the chain was 1 milliseconds

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

3

View the Blockchain

```
[{"index":0,"timestamp":"2023-10-27 15:19:00.02","tx":"Genesis","previousHash":"","nonce":216,"difficulty":2}, {"index":1,"timestamp":"2023-10-27 15:19:12.746","tx":"Mike pays Marty 100 DS", "previousHash":"0073cf07aeacb687592854d2b6b6015c15f4dd8c7d64771c52e910413e59e0da", "nonce":186, "difficulty":2}, {"index":2,"timestamp":"2023-10-27 15:19:31.689","tx":"Marty pays Joe 50 DS", "previousHash":"00615d7e516ddb71f730559bd369940cfedfe38450f11279ff09c030ccb277df", "nonce":44, "difficulty":2}, {"index":3,"timestamp":"2023-10-27 15:19:46.648","tx":"Joe pays Andy 10 DS", "previousHash":"00b69ab8a862c2f5cad161164fb8139dd472cb73ff0f5af0a6f54bf3a4bd4b9d", "nonce":296, "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

Mike pays Marty 76 DS

Block 1 now holds Mike pays Marty 76 DS

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":"2023-10-27 15:19:00.02","tx":"Genesis","previousHash":"","nonce":216,"difficulty":2}, {"index":1,"timestamp":"2023-10-27 15:19:12.746","tx":"Mike pays Marty 76 DS", "previousHash":"0073cf07aeacb687592854d2b6b6015c15f4dd8c7d64771c52e910413e59e0da", "nonce":186, "difficulty":2}, {"index":2,"timestamp":"2023-10-27 15:19:31.689","tx":"Marty pays Joe 50 DS", "previousHash":"00615d7e516ddb71f730559bd369940cfedfe38450f11279ff09c030ccb277df", "nonce":44, "difficulty":2}, {"index":3,"timestamp":"2023-10-27 15:19:46.648","tx":"Joe pays Andy 10 DS", "previousHash":"00b69ab8a862c2f5cad161164fb8139dd472cb73ff0f5af0a6f54bf3a4bd4b9d", "nonce":296, "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 1 milliseconds

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

5

Total execution time 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

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:

Andy pays Sean 25 DSCoin

Total execution time to add this block was 67 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: 3205

Expected total hashes required for the whole chain: 66560.0

Nonce for the most recent block: 21176

Chain hash: 000032b2bed924b362e7ebe6f4adc57498789e9bc9627173f8886acad1ed1e99

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:

Ariane pays Marty 100 DSCoin

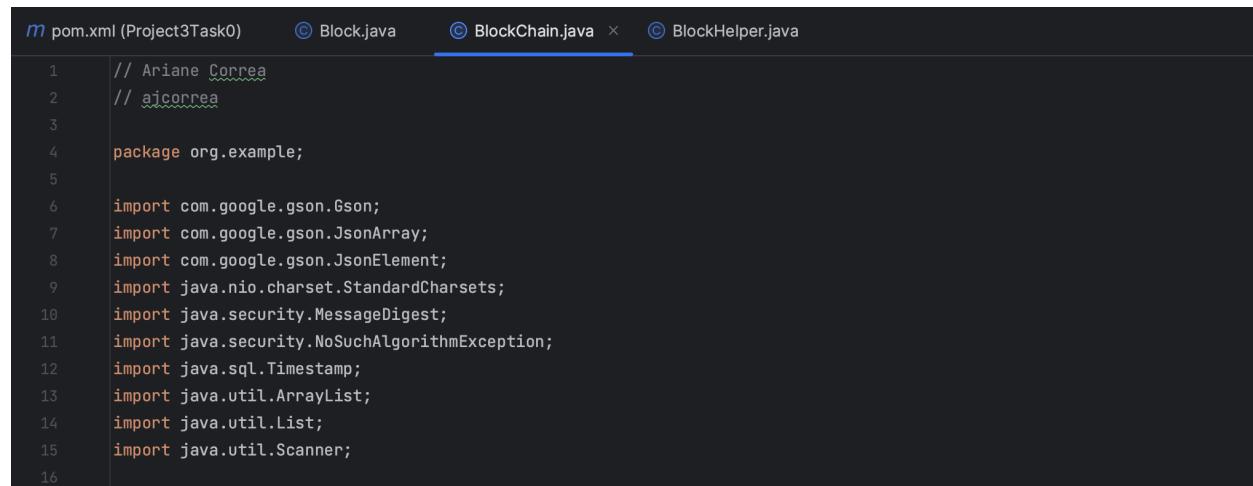
Total execution time to add this block was 564 milliseconds

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

6

Process finished with exit code 0

### Code for BlockChain.java



```
m pom.xml (Project3Task0)   ⚒ Block.java   ⚒ BlockChain.java   ⚒ BlockHelper.java
1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5
6 import com.google.gson.Gson;
7 import com.google.gson.JsonArray;
8 import com.google.gson.JsonElement;
9 import com.google.gson.StandardCharsets;
10 import java.security.MessageDigest;
11 import java.security.NoSuchAlgorithmException;
12 import java.sql.Timestamp;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.Scanner;
```

```
17 > public class BlockChain {
18
19     // Private instance variables
20     28 usages
21     private List<Block> blockList;
22     6 usages
23     private String chainHash;
24     3 usages
25     private int hashesPerSecond;
26     1 usage
27     Gson gson = new Gson();
28
29
30     /**
31      * Constructor to initialize a new blockchain.
32      */
33     1 usage
34     public BlockChain() {
35         this.blockList = new ArrayList<>();
36         this.chainHash = "";
37         this.hashesPerSecond = 0;
38     }
39
40     /**
41      * Get the current blockchain's hash.
42      *
43      * @return The hash of the entire blockchain.
44      */
45     3 usages
46     public String getChainHash() {
47         return chainHash;
48     }
49
50
51     /**
52      * Get the current system timestamp.
53      *
54      * @return A Timestamp object representing the current time.
55      */
56     10 usages
57     public Timestamp getTime() {
58         return new Timestamp(System.currentTimeMillis());
59     }
60
61     /**
62      * Get the most recent block in the blockchain.
63      *
64      * @return The latest Block object in the chain.
65      */
66     2 usages
67     public Block getLatestBlock() {
68         return blockList.get(blockList.size() - 1);
69     }
70 }
```

```

61  /**
62   * Get the size (number of blocks) of the blockchain.
63   *
64   * @return The number of blocks in the blockchain.
65   */
66  2 usages
67  public int getChainSize() {
68      return blockList.size();
69  }
70
71  /**
72   * Compute the number of hash calculations per second.
73   *
74   * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
75   */
76  1 usage
77  public void computeHashesPerSecond() throws NoSuchAlgorithmException {
78      MessageDigest md = MessageDigest.getInstance(algorithm: "SHA-256");
79      int i = 0;
80      Timestamp startTime = getTime();
81      while (i < 2000000) {
82          // Calculate a hash 2,000,000 times with a fixed input
83          byte[] encodedHash = md.digest("00000000".getBytes(StandardCharsets.UTF_8));
84          i += 1;
85      }
86      Timestamp endTime = getTime();
87      this.hashesPerSecond = 2000000 / (int) (endTime.getTime() - startTime.getTime());
88  }
89
90  /**
91   * Get the number of hash calculations per second.
92   *
93   * @return The number of hash calculations per second.
94   */
95  1 usage
96  public int getHashesPerSecond() {
97      return this.hashesPerSecond;
98  }
99
100 /**
101  * Add a new block to the blockchain.
102  *
103  * @param newBlock The Block object to be added.
104  * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
105  */
106  2 usages
107  public void addBlock(Block newBlock) throws NoSuchAlgorithmException {
108      if (this.blockList.size() > 0)
109          newBlock.setPreviousHash(getChainHash());
110      else
111          newBlock.setPreviousHash("");
112      blockList.add(newBlock);
113      this.chainHash = newBlock.proofOfWork();
114  }

```

```

112 /**
113  * Main method to interact with the blockchain.
114 *
115 *For difficulty = 2
116 * Add block 1: 3 ms
117 * Add block 2: 4 ms
118 * Add block 3: 3 ms
119 * Verify: 2 ms
120 * Repair: 17 ms
121 *
122 * For difficulty = 3
123 * Add block 1: 32 ms
124 * Add block 2: 33 ms
125 * Add block 3: 3 ms
126 * Verify: 1 ms
127 * Repair: 13 ms
128 *
129 * For difficulty = 4
130 * Add block 1: 156 ms
131 * Add block 2: 86 ms
132 * Add block 3: 50 ms
133 * Verify: 1 ms
134 * Repair: 400 ms
135 *
136 * For difficulty = 5
137 * Add block 1: 808 ms
138 * Add block 2: 2894 ms
139 * Add block 3: 1274 ms
140 * Verify: 2 ms
141 * Repair: 3507 ms
142 *
143 * As the difficulty level increases, the time required to add new blocks grows substantially.
144 * This is expected because finding a valid nonce becomes more computationally intensive.
145 *
146 * Verifying the chain remains relatively fast and consistent, regardless of the difficulty level.
147 * Verifying the proof of work in existing blocks doesn't become significantly harder as the difficulty increases.
148 *
149 * Repairing the chain becomes increasingly time-consuming with higher difficulty levels.
150 * Recomputing the proof of work for each block, especially for a longer chain, can be a resource-intensive task.
151 *
152 * The repair operation time is directly affected by the chain length.
153 * As the chain grows, the time to repair it also increases.
154 *
155 * These results illustrate the trade-off between security and performance in a blockchain system.
156 * Higher difficulty levels improve security by making it more difficult for malicious actors to alter the chain.
157 * However, this increased security comes at the cost of longer block creation and chain repair times.
158 *
159 * @param args Command-line arguments (not used).
160 * @throws NoSuchAlgorithmException If SHA-256 algorithm is not available.
161 */
162 ▷ public static void main(String args[]) throws NoSuchAlgorithmException {
163     // Create a new blockchain instance
164     BlockChain chain = new BlockChain();
165

```

```

166     // Adding genesis block to the chain
167     Block genesis = new Block(index: 0, chain.getTime(), data: "Genesis", difficulty: 2);
168     genesis.setPreviousHash("");
169     genesis.proofOfWork();
170     chain.computeHashesPerSecond();
171     chain.addBlock(genesis);
172
173     // Create a scanner for user input
174     Scanner sc = new Scanner(System.in);
175     int option = 0;
176
177     while (true) {
178         System.out.println("0. View basic blockchain status.\n" +
179             "1. Add a transaction to the blockchain.\n" +
180             "2. Verify the blockchain.\n" +
181             "3. View the blockchain.\n" +
182             "4. Corrupt the chain.\n" +
183             "5. Hide the corruption by repairing the chain.\n" +
184             "6. Exit");
185         option = sc.nextInt();
186         switch (option) {
187             case 0: {
188                 // Status of blockchain
189                 System.out.println("Current size of chain: " + chain.getChainSize());
190                 System.out.println("Difficulty of most recent block: " + chain.getLatestBlock().getDifficulty());
191                 System.out.println("Total difficulty for all blocks: " + chain.getTotalDifficulty());
192                 System.out.println("Approximate hashes per second " +
193                     "on this machine: " + chain.getHashesPerSecond());
194                 System.out.println("Expected total hashes required for " +
195                     "the whole chain: " + chain.getTotalExpectedHashes());
196                 System.out.println("Nonce for the most recent block: " + chain.getLatestBlock().getNonce());
197                 System.out.println("Chain hash: " + chain.getChainHash() + "\n");
198                 break;
199             }
200             // Adding block to blockchain
201             case 1: {
202                 System.out.println("Enter difficulty > 0 \n");
203                 int difficulty = sc.nextInt();
204                 sc.nextLine();
205                 System.out.println("Enter transaction: \n");
206                 String transaction = sc.nextLine();
207                 Timestamp start = chain.getTime();
208                 // Add new block to the chain
209                 Block newBlock = new Block(chain.getChainSize(), chain.getTime(), transaction, difficulty);
210                 newBlock.setPreviousHash(chain.getChainHash());
211                 newBlock.proofOfWork();
212                 chain.addBlock(newBlock);
213                 Timestamp end = chain.getTime();
214                 System.out.println("Total execution time to add this block was " +
215                     (end.getTime() - start.getTime()) + " milliseconds");
216                 break;
217             }
218         }
219     }

```

```
220 // Verify blockchain
221 case 2: {
222     Timestamp start = chain.getTime();
223     System.out.println("Chain verification: " + chain.isChainValid());
224     Timestamp end = chain.getTime();
225     System.out.println("Total execution time to verify the chain was " +
226                         [(end.getTime() - start.getTime()) + " milliseconds"]);
227     break;
228 }
229
230 // View blockchain
231 case 3: {
232     System.out.println("View the Blockchain");
233     System.out.println(chain.toString());
234     break;
235 }
236
237 // Corrupt blockchain
238 case 4: {
239     System.out.println("corrupt the Blockchain");
240     System.out.println("Enter block ID of block to corrupt");
241     int index = sc.nextInt();
242     sc.nextLine();
243     System.out.println("Enter new data for block " + index);
244     String transaction = sc.nextLine();
245     // Setting data to the selected block in the chain
246     chain.getBlock(index).setData(transaction);
247     System.out.println("Block " + index + " now holds " + transaction);
248     break;
249 }
250
251 // Repair blockchain
252 case 5: {
253     Timestamp start = chain.getTime();
254     chain.repairChain();
255     Timestamp end = chain.getTime();
256     System.out.println("Total execution time required to repair the chain was " +
257                         [(end.getTime() - start.getTime()) + " milliseconds"]);
258     break;
259 }
260
261 // Exit the program
262 case 6: {
263     System.exit( status: 0 );
264 }
265 }
266 }
267 }
```

```
268     /**
269      * Converts the blockchain to a JSON representation.
270      *
271      * @return JSON string representing the blockchain.
272      */
273     @Override
274     public String toString() {
275         JSONArray jsonArray = new JSONArray();
276         for (Block b : blockList) {
277             jsonArray.add(json.Gson.toJson(b.toString(), JsonElement.class));
278         }
279         return jsonArray.toString();
280     }
281
282     /**
283      * Get a specific block from the blockchain based on its index.
284      *
285      * @param i The index of the block to retrieve.
286      * @return The block at the specified index.
287      */
288     > public Block getBlock(int i) { return blockList.get(i); }
289
290     /**
291      * Calculate the total difficulty of all blocks in the blockchain.
292      *
293      * @return The total difficulty of the blockchain.
294      */
295     > public int getTotalDifficulty() {
296         int totalDifficulty = 0;
297         for (Block block : blockList) {
298             totalDifficulty += block.getDifficulty();
299         }
300         return totalDifficulty;
301     }
302
303     /**
304      * Calculate the total expected hash calculations required for the entire blockchain.
305      *
306      * @return The total expected hash calculations needed for the blockchain.
307      */
308     > public double getTotalExpectedHashes() {
309         double totalExpectedHashes = 0;
310         for (Block block : blockList)
311             totalExpectedHashes += Math.pow(16, block.getDifficulty()); // 16 (16 hex characters) ^ difficulty of block
312         return totalExpectedHashes;
313     }
314
315 }
```

```
316
317     /**
318      * Check the validity of the blockchain.
319      *
320      * @return "TRUE" if the blockchain is valid; "FALSE" with an explanation otherwise.
321      */
322
323     1usage
324     public String isChainValid() throws NoSuchAlgorithmException {
325         // Chain contains only 1 block , i.e. genesis
326         if (blockList.size() == 1) {
327             Block genesisBlock = this.blockList.get(0);
328             String hash = genesisBlock.calculateHash();
329             // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
330             String prefix = new String(new char[genesisBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
331             if (!hash.substring(0, genesisBlock.getDifficulty()).equals(prefix)) {
332                 return "FALSE \n Improper hash on genesis node";
333             } else if (!chainHash.equals(hash)) {
334                 return "FALSE \n Chain hash and computed hash do not match";
335             } else {
336                 return "TRUE";
337             }
338
339             // Chain contains more than 1 block
340             if (blockList.size() > 1) {
341                 for (int i = 1; i < blockList.size(); i++) {
342                     Block currentBlock = this.blockList.get(i);
343                     Block previousBlock = this.blockList.get(i - 1);
344                     String hash = currentBlock.calculateHash();
345                     String hashPointer = currentBlock.getPreviousHash();
346                     // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
347                     String prefix = new String(new char[currentBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
348                     if (!hash.substring(0, currentBlock.getDifficulty()).equals(prefix))
349                         return "FALSE \n Improper hash on node " + i + " Does not begin with " + prefix;
350                     // Check proof of work / leading zeros
351                     else if (!hashPointer.equals(previousBlock.calculateHash()))
352                         return "FALSE \n Improper previous hash";
353                 }
354
355                 // A chain hash, check the last element added to to the blocklist
356                 if (!chainHash.equals(blockList.get(blockList.size() - 1).calculateHash())) {
357                     return "Chain hash error";
358                 }
359
360                 return "TRUE";
361             }
362         }
```

```

1 usage
363     public void repairChain() throws NoSuchAlgorithmException {
364         // Genesis block
365         if (blockList.size() == 1) {
366             // Reset previous hash and recompute proof of work
367             blockList.get(0).setPreviousHash("");
368             blockList.get(0).proofOfWork();
369         }
370
371         if (blockList.size() > 1) {
372             for (int i = 1; i < blockList.size(); i++) {
373                 // Reset previous hash and recompute proof of work
374                 blockList.get(i).setPreviousHash(blockList.get(i - 1).calculateHash());
375                 blockList.get(i).proofOfWork();
376             }
377
378             // Reset chain hash
379             this.chainHash = blockList.get(blockList.size() - 1).calculateHash();
380         }
381     }
382 }
383

```

## Code for Block.java

```

m pom.xml (Project3Task0)  ⌂ Block.java ✘  ⌂ BlockChain.java  ⌂ BlockHelper.java
1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5
6 import com.google.gson.JsonObject;
7 import java.math.BigInteger;
8 import java.nio.charset.StandardCharsets;
9 import java.security.MessageDigest;
10 import java.security.NoSuchAlgorithmException;
11 import java.sql.Timestamp;
12 usages
13
14 public class Block {
15
16     // Stores the position of the block on the chain
17     private int index;
18
19     // Stores the timestamp of the instant when the block was created
20     private Timestamp timestamp;
21
22     // Stores single transaction details of the block
23     private String data;
24
25     // Stores the SHA256 hash of the block's parent
26     private String previousHash;
27
28     // Value determined by POW (Proof of Work) routine
29     private BigInteger nonce;
30
31     // The minimum number of leftmost hex digits needed by a proper hash
32     private int difficulty;
33
34

```

```

27  /**
28   * Constructor for creating a Block object.
29   *
30   * @param index      The position of the block on the chain.
31   * @param timestamp  The timestamp of when the block was created.
32   * @param data        Single transaction details of the block.
33   * @param difficulty The difficulty level for proof of work.
34   */
35  2 usages
36  public Block(int index, Timestamp timestamp, String data, int difficulty) {
37      this.index = index;
38      this.timestamp = timestamp;
39      this.data = data;
40      this.difficulty = difficulty;
41      this.nonce = BigInteger.ZERO;
42  }
43
44  /**
45   * Calculate and return the SHA-256 hash of the block.
46   *
47   * @return The calculated SHA-256 hash.
48   * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
49   */
50  8 usages
51  public String calculateHash() throws NoSuchAlgorithmException {
52      String parentString = String.valueOf(this.index) + this.timestamp + this.data +
53          this.previousHash + this.nonce + this.difficulty;
54      MessageDigest md = MessageDigest.getInstance(algorithm: "SHA-256");
55      byte[] encodedHash = md.digest(parentString.getBytes(StandardCharsets.UTF_8));
56      return BlockHelper.bytesToHex(encodedHash);
57  }
💡
58  /**
59   * Get the current nonce value.
60   *
61   * @return The nonce value.
62   */
63  1 usage
64  public BigInteger getNonce() {
65      return nonce;
66  }
67
68  /**
69   * Get the difficulty level for proof of work.
70   *
71   * @return The difficulty level.
72   */
73  7 usages
74  public int getDifficulty() {
75      return difficulty;
76  }
77
78  /**
79   * Set the difficulty level for proof of work.
80   *
81   * @param difficulty The new difficulty level.
82   */
83  no usages
84  public void setDifficulty(int difficulty) {
85      this.difficulty = difficulty;
86  }

```

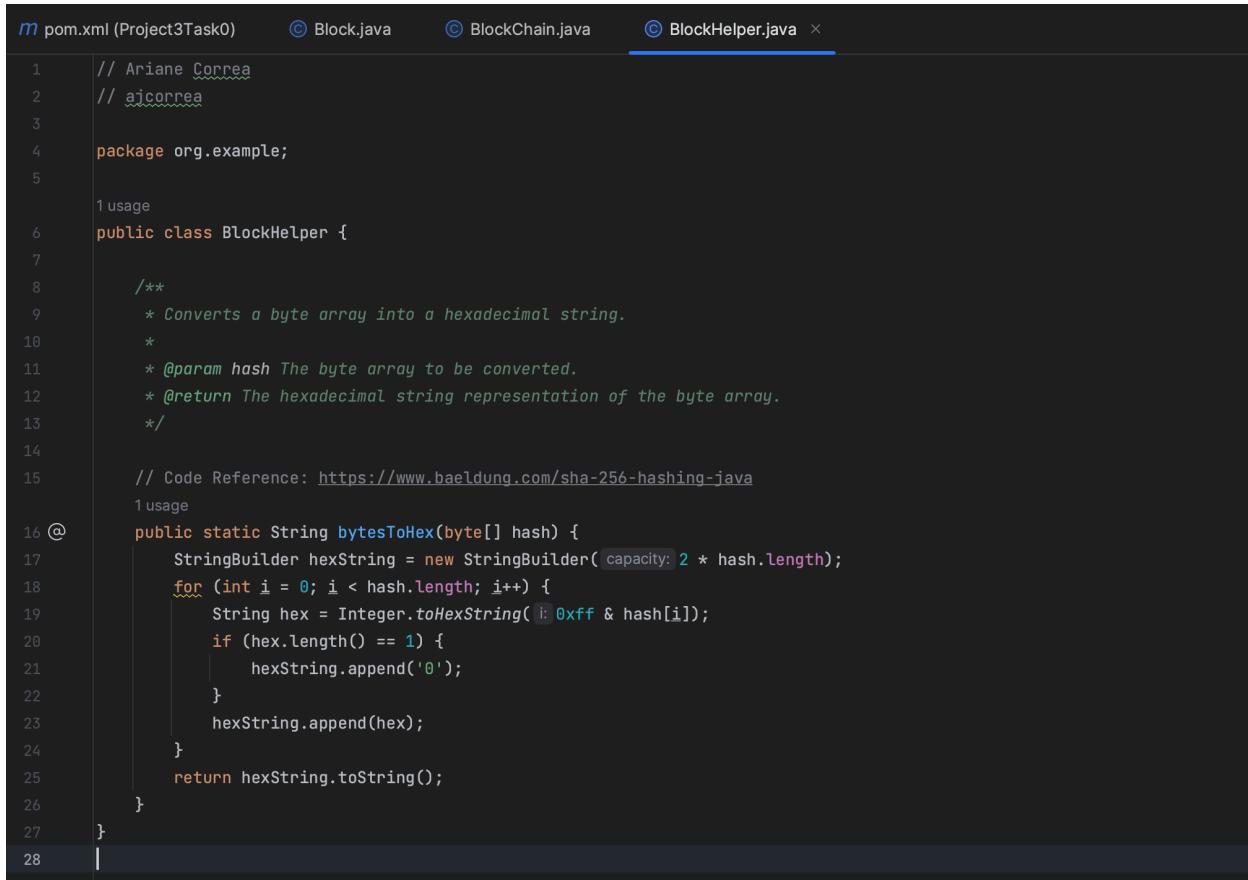
```
84     /**
85      * Get the position of the block on the chain.
86      *
87      * @return The block's index.
88      */
89     no usages
90     public int getIndex() {
91         return index;
92     }
93
94     /**
95      * Get the timestamp when the block was created.
96      *
97      * @return The timestamp of block creation.
98      */
99     no usages
100    public Timestamp getTimestamp() {
101        return timestamp;
102    }
103
104    /**
105     * Get the data (transaction details) of the block.
106     *
107     * @return The block's data.
108     */
109     no usages
110    public String getData() {
111        return data;
112    }
113
114    /**
115     * Get the SHA-256 hash of the block's parent.
116     *
117     * @return The SHA-256 hash of the parent block.
118     */
119     1 usage
120    public String getPreviousHash() {
121        return previousHash;
122    }
123
124    /**
125     * Set the position of the block on the chain.
126     *
127     * @param index The new block index.
128     */
129     no usages
130    public void setIndex(int index) {
131        this.index = index;
132    }
133
134    /**
135     * Set the timestamp when the block was created.
136     *
137     * @param timestamp The new block creation timestamp.
138     */
139     no usages
140    public void setTimestamp(Timestamp timestamp) {
141        this.timestamp = timestamp;
142    }
143
```

```

138     /**
139      * Set the data (transaction details) of the block.
140      *
141      * @param data The new block data.
142      */
143
144     1 usage
145     public void setData(String data) {
146         this.data = data;
147     }
148
149     /**
150      * Set the SHA-256 hash of the block's parent.
151      *
152      * @param previousHash The new parent block's hash.
153      */
154
155     6 usages
156     public void setPreviousHash(String previousHash) {
157         this.previousHash = previousHash;
158     }
159
160     /**
161      * Convert the block's attributes to a JSON representation.
162      *
163      * @return A JSON string representing the block's attributes.
164      */
165     @Override
166     public String toString() {
167         JSONObject jsonObject = new JSONObject();
168         jsonObject.addProperty( property: "index", index);
169         jsonObject.addProperty( property: "timestamp", String.valueOf(timestamp));
170         jsonObject.addProperty( property: "tx", data);
171         jsonObject.addProperty( property: "previousHash", previousHash);
172         jsonObject.addProperty( property: "nonce", nonce);
173         jsonObject.addProperty( property: "difficulty", difficulty);
174         return jsonObject.toString();
175     }
176
177     /**
178      * Perform Proof of Work (POW) to find a hash that meets the required difficulty level.
179      *
180      * @return The valid SHA-256 hash after POW.
181      * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
182      */
183
184     5 usages
185     public String proofOfWork() throws NoSuchAlgorithmException {
186         String hexHash = calculateHash();
187         String matchString = "";
188         for (int i = 0; i < this.difficulty; i++) {
189             matchString = matchString + "0";
190         }
191         while (!hexHash.substring(0, this.difficulty).equalsIgnoreCase(matchString)) {
192             this.nonce = this.nonce.add(BigInteger.ONE);
193             hexHash = calculateHash();
194         }
195         return hexHash;
196     }
197 }

```

## Code for BlockHelper.java



```
1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5
6 1 usage
7
8 /**
9 * Converts a byte array into a hexadecimal string.
10 *
11 * @param hash The byte array to be converted.
12 * @return The hexadecimal string representation of the byte array.
13 */
14
15 // Code Reference: https://www.baeldung.com/sha-256-hashing-java
16 1 usage
17 @
18     public static String bytesToHex(byte[] hash) {
19         StringBuilder hexString = new StringBuilder( capacity: 2 * hash.length);
20         for (int i = 0; i < hash.length; i++) {
21             String hex = Integer.toHexString( i: 0xff & hash[i]);
22             if (hex.length() == 1) {
23                 hexString.append('0');
24             }
25             hexString.append(hex);
26         }
27     }
28 }
```

## Project 3

Name: Ariane Correa  
Andrew ID: ajcorrea

### Task 1

#### **Console Output (Server Side):**

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -  
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=54059:/Applications/IntelliJ  
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath  
/Users/arianecorrea/IdeaProjects/Project3Task1/target/classes:/Users/arianecorrea/.m2/repos  
itory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.EchoServerTCP  
Blockchain server running
```

We have a visitor

Response :

```
{"selection":0,"size":1,"chainHash":"0004da177de7c5403c5afff3b1e0f309b62e8fdec916a2e1a  
1d19b06874fa8b","totalHashes":256.0,"totalDiff":2,"recentNonce":387,"diff":2,"hps":2000000}
```

Adding a block

```
Setting response to : {"selection":1,"response":"Total execution time to add this block was 11  
milliseconds"}
```

Adding a block

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

Adding a block

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

Verifying entire chain

Chain verification: TRUE

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

View the Blockchain

```
Setting response to: {"selection":3,"response":"[{"index":0,"timestamp":"2023-10-27  
19:08:10.476","tx":"Genesis","previousHash":"","nonce":387,"difficulty":2},  
{"index":1,"timestamp":"2023-10-27 19:08:54.805","tx":"Mike pays Marty 100  
DSCoin","previousHash":"0004da177de7c5403c5afff3b1e0f309b62e8fdec916a2e1a1d19b  
06874fa8b","nonce":278,"difficulty":2},  
{"index":2,"timestamp":"2023-10-27  
19:09:14.847","tx":"Marty pays Joe 50"}]
```

DSCoin\" , "previousHash": \"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c3fd6545d1\" , "nonce": 355, "difficulty": 2}, {"index": 3, "timestamp": "2023-10-27 19:09:29.694"}, {"tx": "Joe pays Andy 10"}  
DSCoin\" , "previousHash": \"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398e351a2cc1b\" , "nonce": 236, "difficulty": 2}]}  
Corrupt the Blockchain  
Setting response to: {"selection": 4, "response": "Block 1 now holds Mike pays Marty 76 DSCoin"}  
View the Blockchain  
Setting response to: {"selection": 3, "response": "[{"index": 0, "timestamp": "2023-10-27 19:08:10.476"}, {"tx": "Genesis", "previousHash": "", "nonce": 387, "difficulty": 2}, {"index": 1, "timestamp": "2023-10-27 19:08:54.805"}, {"tx": "Mike pays Marty 76"}]  
DSCoin\" , "previousHash": \"0004da177de7c5403c5afff3b1e0f309b62e8fdec916a2e1a1d19b06874fa8b\" , "nonce": 278, "difficulty": 2}, {"index": 2, "timestamp": "2023-10-27 19:09:14.847"}, {"tx": "Marty pays Joe 50"}  
DSCoin\" , "previousHash": \"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c3fd6545d1\" , "nonce": 355, "difficulty": 2}, {"index": 3, "timestamp": "2023-10-27 19:09:29.694"}, {"tx": "Joe pays Andy 10"}  
DSCoin\" , "previousHash": \"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398e351a2cc1b\" , "nonce": 236, "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 2 milliseconds"}  
Repairing the entire chain  
Setting response to: {"selection": 5, "response": "Total execution time required to repair the chain was 11 milliseconds"}  
Verifying entire chain  
Chain verification: TRUE  
Setting response to: {"selection": 2, "response": "Total execution time to verify the chain was 2 milliseconds"}  
Adding a block  
Setting response to : {"selection": 1, "response": "Total execution time to add this block was 166 milliseconds"}  
Response :  
{"selection": 0, "size": 5, "chainHash": "00006717d60d5f7601c13ca24c43cfef0078dc2ce9cc978ba444b024f9f81a7b", "totalHashes": 66560.0, "totalDiff": 12, "recentNonce": 88571, "diff": 4, "hps": 200000}  
Adding a block  
Setting response to : {"selection": 1, "response": "Total execution time to add this block was 1674 milliseconds"}  
Process finished with exit code 0

### **Console Output (Client Side):**

```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -  
javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=54063:/Applications/IntelliJ  
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath  
/Users/arianecorrea/IdeaProjects/Project3Task1/target/classes:/Users/arianecorrea/.m2/repos  
itory/com/google/code/gson/gson/2.9.0/gson-2.9.0.jar org.example.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: 387

Chain hash: "0004da177de7c5403c5afff3b1e0f309b62e8fdec916a2e1a1d19b06874fa8b"

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:

Mike pays Marty 100 DSCoin

"Total execution time to add this block was 11 milliseconds"

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

1

Enter difficulty > 0

2

Enter transaction:

Marty pays Joe 50 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:

Joe pays Andy 10 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

2

"Total execution time to verify the chain was 2 milliseconds"

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.

6. Exit

3

```
[{"index":0,"timestamp":"2023-10-27  
19:08:10.476","tx":"Genesis","previousHash": "", "nonce":387,"difficulty":2}, {"index":1,"timestamp":"2023-10-27 19:08:54.805","tx":"Mike pays Marty 100  
DSCoin","previousHash": "0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b  
06874fa8b","nonce":278,"difficulty":2}, {"index":2,"timestamp":"2023-10-27  
19:09:14.847","tx":"Marty pays Joe 50  
DSCoin","previousHash": "00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c  
3fd6545d1","nonce":355,"difficulty":2}, {"index":3,"timestamp":"2023-10-27  
19:09:29.694","tx":"Joe pays Andy 10  
DSCoin","previousHash": "00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398  
e351a2cc1b","nonce":236,"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

Mike pays Marty 76 DSCoin

"Block 1 now holds Mike pays Marty 76 DSCoin"

0. View basic blockchain status.

1. Add a transaction to the blockchain.

2. Verify the blockchain.

3. View the blockchain.

4. Corrupt the chain.

5. Hide the corruption by repairing the chain.

6. Exit

3

```
[{"index":0,"timestamp":"2023-10-27  
19:08:10.476","tx":"Genesis","previousHash": "", "nonce":387,"difficulty":2}, {"index":1,"timestamp":"2023-10-27 19:08:54.805","tx":"Mike pays Marty 76  
DSCoin","previousHash": "0004da177de7c5403c5afff3b1e0f309b62e8fdecb916a2e1a1d19b
```

```
06874fa8b\" , \"nonce\":278, \"difficulty\":2}, {\"index\":2, \"timestamp\": \"2023-10-27  
19:09:14.847\", \"tx\": \"Marty pays Joe 50  
DSCoin\", \"previousHash\": \"00502d758a21723f5338c47cbd75deb2e096efca4cb5198d53bf47c  
3fd6545d1\", \"nonce\":355, \"difficulty\":2}, {\"index\":3, \"timestamp\": \"2023-10-27  
19:09:29.694\", \"tx\": \"Joe pays Andy 10  
DSCoin\", \"previousHash\": \"00c82c9ed25bd311b358aec3610f33fea73079fa75b582960a9398  
e351a2cc1b\", \"nonce\":236, \"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 2 milliseconds"

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

5

"Total execution time required to repair the chain was 11 milliseconds"

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.
3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

2

"Total execution time to verify the chain was 2 milliseconds"

0. View basic blockchain status.
1. Add a transaction to the blockchain.
2. Verify the blockchain.

3. View the blockchain.
4. Corrupt the chain.
5. Hide the corruption by repairing the chain.
6. Exit

1

Enter difficulty > 0

4

Enter transaction:

Andy pays Sean 25 DSCoin

"Total execution time to add this block was 166 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: 88571

Chain hash: "00006717d60d5f7601c13ca24c43cfef0078dc2ce9cc978ba444b024f9f81a7b"

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:

Ariane pays Marty 100 DSCoin

"Total execution time to add this block was 1674 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

6

Process finished with exit code 0

### Code for Block.java

```
© Block.java x © BlockChain.java © BlockHelper.java © EchoClientTCP.java © EchoServerTCP.java © RequestMessage.java
1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5
6 import com.google.gson.JsonObject;
7 import java.math.BigInteger;
8 import java.nio.charset.StandardCharsets;
9 import java.security.MessageDigest;
10 import java.security.NoSuchAlgorithmException;
11 import java.sql.Timestamp;
12
13 14 usages
14 public class Block {
15     // Stores the position of the block on the chain
16     5 usages
17     private int index;
18     // Stores the timestamp of the instant when the block was created
19     5 usages
20     private Timestamp timestamp;
21     // Stores single transaction details of the block
22     5 usages
23     private String data;
24     // Stores the SHA256 hash of the block's parent
25     4 usages
26     private String previousHash;
27     // Value determined by POW (Proof of Work) routine
28     6 usages
29     private BigInteger nonce;
30     // The minimum number of leftmost hex digits needed by a proper hash
31     7 usages
32     private int difficulty;
```

```

27  /**
28   * Constructor for creating a Block object.
29   *
30   * @param index      The position of the block on the chain.
31   * @param timestamp  The timestamp of when the block was created.
32   * @param data        Single transaction details of the block.
33   * @param difficulty The difficulty level for proof of work.
34   */
35  2 usages
36  public Block(int index, Timestamp timestamp, String data, int difficulty) {
37      this.index = index;
38      this.timestamp = timestamp;
39      this.data = data;
40      this.difficulty = difficulty;
41      this.nonce = BigInteger.ZERO;
42  }
43
44  /**
45   * Calculate and return the SHA-256 hash of the block.
46   *
47   * @return The calculated SHA-256 hash.
48   * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
49   */
50  8 usages
51  public String calculateHash() throws NoSuchAlgorithmException {
52      String parentString = String.valueOf(this.index) + this.timestamp + this.data +
53          this.previousHash + this.nonce + this.difficulty;
54      MessageDigest md = MessageDigest.getInstance(algorithm: "SHA-256");
55      byte[] encodedHash = md.digest(
56          parentString.getBytes(StandardCharsets.UTF_8));
57      return BlockHelper.bytesToHex(encodedHash);
58  }
59
60  /**
61   * Get the current nonce value.
62   *
63   * @return The nonce value.
64   */
65  1 usage
66  public BigInteger getNonce() {
67      return nonce;
68  }
69
70  /**
71   * Get the difficulty level for proof of work.
72   *
73   * @return The difficulty level.
74   */
75  7 usages
76  public int getDifficulty() {
77      return difficulty;
78  }
79
80  /**
81   * Simple setter method
82   *
83   * @param difficulty - determines how much work is required to produce a proper hash
84   */
85  no usages
86  public void setDifficulty(int difficulty) {
87      this.difficulty = difficulty;
88  }

```

```
85  /**
86   * Get the position of the block on the chain.
87   *
88   * @return The block's index.
89   */
90 no usages
91 public int getIndex() {
92     return index;
93 }
94
95 /**
96  * Get the timestamp when the block was created.
97  *
98  * @return The timestamp of block creation.
99  */
100 no usages
101 public Timestamp getTimestamp() {
102     return timestamp;
103 }
104
105 /**
106  * Get the data (transaction details) of the block.
107  *
108  * @return The block's data.
109  */
110 no usages
111 public String getData() {
112     return data;
113 }
114
115 /**
116  * Get the SHA-256 hash of the block's parent.
117  *
118  * @return The SHA-256 hash of the parent block.
119  */
120
121 /**
122  * Set the position of the block on the chain.
123  *
124  * @param index The new block index.
125  */
126 no usages
127 public void setIndex(int index) {
128     this.index = index;
129 }
130
131 /**
132  * Set the timestamp when the block was created.
133  *
134  * @param timestamp The new block creation timestamp.
135  */
136 no usages
137 public void setTimestamp(Timestamp timestamp) {
138     this.timestamp = timestamp;
139 }
```

```

139     /**
140      * Set the data (transaction details) of the block.
141      *
142      * @param data The new block data.
143      */
144     1 usage
145     public void setData(String data) {
146         this.data = data;
147     }
148
149     /**
150      * Set the SHA-256 hash of the block's parent.
151      *
152      * @param previousHash The new parent block's hash.
153      */
154     6 usages
155     public void setPreviousHash(String previousHash) {
156         this.previousHash = previousHash;
157     }
158
159     /**
160      * Convert the block's attributes to a JSON representation.
161      *
162      * @return A JSON string representing the block's attributes.
163      */
164     @Override
165     public String toString() {
166         JsonObject jsonObject = new JsonObject();
167         jsonObject.addProperty( property: "index", index);
168         jsonObject.addProperty( property: "timestamp", String.valueOf(timestamp));
169         jsonObject.addProperty( property: "tx", data);
170         jsonObject.addProperty( property: "previousHash", previousHash);
171         jsonObject.addProperty( property: "nonce", nonce);
172         jsonObject.addProperty( property: "difficulty", difficulty);
173         return jsonObject.toString();
174     }
175
176     /**
177      * Perform Proof of Work (POW) to find a hash that meets the required difficulty level.
178      *
179      * @return The valid SHA-256 hash after POW.
180      * @throws NoSuchAlgorithmException If SHA-256 hashing algorithm is not available.
181      */
182     5 usages
183     public String proofOfWork() throws NoSuchAlgorithmException {
184
185         String hexHash = calculateHash();
186         String matchString = "";
187         for (int i = 0; i < this.difficulty; i++) {
188             matchString = matchString + "0";
189         }
190         while(!hexHash.substring(0, this.difficulty).equalsIgnoreCase(matchString)){
191             this.nonce = this.nonce.add(BigInteger.ONE);
192             hexHash = calculateHash();
193         }
194         return hexHash;
195     }

```

## Code for BlockChain.java

```
© Block.java © BlockChain.java x © BlockHelper.java © EchoClientTCP.java © EchoServerTCP.java © RequestMessage.java
1 // Ariane Correa
2 // aicorrea
3
4 package org.example;
5
6 import com.google.gson.Gson;
7 import com.google.gson.JsonArray;
8 import com.google.gson.JsonElement;
9 import java.nio.charset.StandardCharsets;
10 import java.security.MessageDigest;
11 import java.security.NoSuchAlgorithmException;
12 import java.sql.Timestamp;
13 import java.util.ArrayList;
14 import java.util.List;
15
16 2 usages
17 public class BlockChain {
18
19     // Private instance variables
20     28 usages
21     private List<Block> blockList;
22     6 usages
23     private String chainHash;
24     3 usages
25     private int hashesPerSecond;
26     1 usage
27     Gson gson = new Gson();
28
29     /**
30      * Constructor to initialize a new blockchain.
31      */
32     1 usage
33     public BlockChain() {
34         this.blockList = new ArrayList<>();
35         this.chainHash = "";
36         this.hashesPerSecond = 0;
37     }
38
39     /**
40      * Get the current blockchain's hash.
41      *
42      * @return The hash of the entire blockchain.
43      */
44     3 usages
45     public String getChainHash() {
46         return chainHash;
47     }
48
49     /**
50      * Get the current system timestamp.
51      *
52      * @return A Timestamp object representing the current time.
53      */
54     10 usages
55     public Timestamp getTime() { return new Timestamp(System.currentTimeMillis()); }
```



```
112     /**
113      * Converts the blockchain to a JSON representation.
114      *
115      * @return JSON string representing the blockchain.
116      */
117     @Override
118     public String toString() {
119
120         JSONArray jsonArray = new JSONArray();
121
122         for (Block b : blockList) {
123             jsonArray.add(Json.fromJson(b.toString(), JsonElement.class));
124         }
125
126         return jsonArray.toString();
127     }
128
129
130     /**
131      * Get a specific block from the blockchain based on its index.
132      *
133      * @param i The index of the block to retrieve.
134      * @return The block at the specified index.
135      */
136     public Block getBlock(int i) {
137         return blockList.get(i);
138     }
139
140
141     /**
142      * Calculate the total difficulty of all blocks in the blockchain.
143      *
144      * @return The total difficulty of the blockchain.
145      */
146     public int getTotalDifficulty() {
147         int totalDifficulty = 0;
148         for (Block block : blockList) {
149             totalDifficulty += block.getDifficulty();
150         }
151         return totalDifficulty;
152     }
153
154     /**
155      * Calculate the total expected hash calculations required for the entire blockchain.
156      *
157      * @return The total expected hash calculations needed for the blockchain.
158      */
159     public double getTotalExpectedHashes() {
160         double totalExpectedHashes = 0;
161         for (Block block : blockList)
162             totalExpectedHashes += Math.pow(16, block.getDifficulty()); // 16 (16 hex characters) ^ difficulty of block
163         return totalExpectedHashes;
164     }
165 }
```

```

166     /**
167      * Check the validity of the blockchain.
168      *
169      * @return "TRUE" if the blockchain is valid; "FALSE" with an explanation otherwise.
170      */
171
172     1usage
173
174     public String isChainValid() throws NoSuchAlgorithmException {
175
176         // Chain contains only 1 block , i.e. genesis
177         if (blockList.size() == 1) {
178             Block genesisBlock = this.blockList.get(0);
179             String hash = genesisBlock.calculateHash();
180
181             // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
182             String prefix = new String(new char[genesisBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
183             if (!hash.substring(0, genesisBlock.getDifficulty()).equals(prefix)) {
184                 return "FALSE \n Improper hash on genesis node";
185             } else if (!chainHash.equals(hash)) {
186                 return "FALSE \n Chain hash and computed hash do not match";
187             } else {
188                 return "TRUE";
189             }
190
191         }
192
193         // More than 1 block
194         if (blockList.size() > 1) {
195             for (int i = 1; i < blockList.size(); i++) {
196                 Block currentBlock = this.blockList.get(i);
197                 Block previousBlock = this.blockList.get(i - 1);
198
199                 String hash = currentBlock.calculateHash();
200                 String hashPointer = currentBlock.getPreviousHash();
201
202                 // Calculate prefix based on difficulty, number of leading zeroes based on the difficulty value
203                 String prefix = new String(new char[currentBlock.getDifficulty()]).replace( target: "\0", replacement: "0");
204
205
206                 if (!hash.substring(0, currentBlock.getDifficulty()).equals(prefix))
207                     return "FALSE \n Improper hash on node " + i + " Does not begin with " + prefix;
208                 // Check proof of work / leading zeros
209                 else if (!hashPointer.equals(previousBlock.calculateHash()))
210                     return "FALSE \n Improper previous hash";
211             }
212
213         }
214
215         // Chain hash , check the last element added to the blocklist
216         if (!chainHash.equals(blockList.get(blockList.size() - 1).calculateHash())) {
217             return "Chain hash error";
218         }
219
220         return "TRUE";
221     }
222

```

```
213
214     /**
215      * This routine repairs the chain. It checks the hashes of each block and ensures
216      * that any illegal hashes are recomputed.
217      * After this routine is run, the chain will be valid. The routine does not modify any difficulty values.
218      * It computes new proof of work based on the difficulty specified in the Block.
219      *
220      * @throws NoSuchAlgorithmException
221     */
222     1 usage
223     public void repairChain() throws NoSuchAlgorithmException {
224
225         // Genesis block
226         if (blockList.size() == 1) {
227             //Reset previous hash and recompute proof of work
228             blockList.get(0).setPreviousHash("");
229             blockList.get(0).proofOfWork();
230         }
231
232         if (blockList.size() > 1) {
233             for (int i = 1; i < blockList.size(); i++) {
234                 // Reset previous hash and recompute proof of work
235                 blockList.get(i).setPreviousHash(blockList.get(i - 1).calculateHash());
236                 blockList.get(i).proofOfWork();
237             }
238
239             // Reset chain hash
240             this.chainHash = blockList.get(blockList.size() - 1).calculateHash();
241         }
242     }
243 }
```

## Code for BlockHelper.java

```
1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5
6 1 usage
7
8 /**
9 * Converts a byte array into a hexadecimal string.
10 *
11 * @param hash The byte array to be converted.
12 * @return The hexadecimal string representation of the byte array.
13 */
14
15 // Code Reference: https://www.baeldung.com/sha-256-hashing-java
16 1 usage
17 @
18 public static String bytesToHex(byte[] hash) {
19     StringBuilder hexString = new StringBuilder( capacity: 2 * hash.length);
20     for (int i = 0; i < hash.length; i++) {
21         String hex = Integer.toHexString( i: 0xff & hash[i]);
22         if (hex.length() == 1) {
23             hexString.append('0');
24         }
25         hexString.append(hex);
26     }
27 }
```

## Code for EchoClientTCP.java

```
8 import com.google.gson.JsonObject;
9 import com.google.gson.JsonParser;
10
11 import java.io.*;
12 import java.net.Socket;
13 import java.util.Scanner;
14
15 public class EchoClientTCP {
16     5 usages
17     static Socket clientSocket = null;
18
19     public static void main(String args[]) {
20         // Arguments supply hostname
21         Scanner sc = new Scanner(System.in);
22         try {
23             System.out.println("The TCP client is running.");
24             clientSocket = new Socket( host: "localhost", port: 7777);
```

```
25 // To read from the server
26 BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
27
28 // To write to the server
29 PrintWriter out = new PrintWriter(new BufferedWriter(
30         new OutputStreamWriter(clientSocket.getOutputStream())));
31
32 int option = 0;
33
34 while (true) {
35
36     System.out.println("0. View basic blockchain status.\n" +
37             "1. Add a transaction to the blockchain.\n" +
38             "2. Verify the blockchain.\n" +
39             "3. View the blockchain.\n" +
40             "4. Corrupt the chain.\n" +
41             "5. Hide the corruption by repairing the chain.\n" +
42             "6. Exit");
43
44     option = sc.nextInt();
45
46     switch (option) {
47         // Status of blockchain
48         case 0: {
49             RequestMessage requestMessage = new RequestMessage(option);
50             out.println(requestMessage.getRequestJson());
51             out.flush();
52             String reply = in.readLine();
53             // Parsing response object from server as json object
54             JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
55             System.out.println("Current size of chain: " + jsonReply.get("size"));
56             System.out.println("Difficulty of most recent block: " + jsonReply.get("diff"));
57             System.out.println("Total difficulty for all blocks: " + jsonReply.get("totalDiff"));
58             System.out.println("Approximate hashes per second on this machine : " + jsonReply.get("hps"));
59             System.out.println("Expected total hashes required for the whole chain: " +
60                 jsonReply.get("totalHashes"));
61             System.out.println("Nonce for most recent block: " + jsonReply.get("recentNonce"));
62             System.out.println("Chain hash: " + jsonReply.get("chainHash") + "\n");
63             break;
64         }
65
66         // Adding block to blockchain
67         case 1: {
68             System.out.println("Enter difficulty > 0 \n");
69             int difficulty = sc.nextInt();
70             sc.nextLine();
71             System.out.println("Enter transaction: \n");
72             String transaction = sc.nextLine();
73             RequestMessage requestMessage = new RequestMessage(option, difficulty, transaction);
74             out.println(requestMessage.getRequestJson());
75             out.flush();
76             String reply = in.readLine();
77             // Parsing response object from server as json object
78             JsonObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
79             System.out.println(jsonReply.get("response"));
80             break;
81         }
82     }
83 }
```

```
83
84
85         // Verify, View, and Repair blockchain
86     case 2, 3, 5: {
87
88         RequestMessage requestMessage = new RequestMessage(option);
89         out.println(requestMessage.getRequestJson());
90         out.flush();
91         String reply = in.readLine();
92         // Parsing response object from server as json object
93         JSONObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
94         System.out.println(jsonReply.get("response"));
95         break;
96     }
97
98     // Corrupt blockchain
99     case 4: {
100         System.out.println("corrupt the Blockchain");
101         System.out.println("Enter block ID of block to corrupt");
102         int index = sc.nextInt();
103         sc.nextLine();
104         System.out.println("Enter new data for block " + index);
105         String transaction = sc.nextLine();
106         RequestMessage requestMessage = new RequestMessage(option, index, transaction);
107         out.println(requestMessage.getRequestJson());
108         out.flush();
109         String reply = in.readLine();
110         //Parsing response object from server as json object
111         JSONObject jsonReply = new JsonParser().parse(reply).getAsJsonObject();
112         System.out.println(jsonReply.get("response"));
113         break;
114     }
115
116     // Exit the program
117     case 6: {
118         RequestMessage requestMessage = new RequestMessage(option);
119         out.println(requestMessage.getRequestJson());
120         out.flush();
121         System.exit(status: 0);
122     }
123
124     }
125   }
126 } catch (IOException e) {
127   System.out.println("IO Exception:" + e.getMessage());
128 } finally {
129   try {
130     if (clientSocket != null) {
131       clientSocket.close();
132     }
133   } catch (IOException e) {
134     // Ignore exception on close
135   }
136 }
137 }
138 }
```

## Code for EchoServerTCP.java

```
ckChain.java      © BlockHelper.java      © EchoClientTCP.java      © EchoServerTCP.java ×      © RequestMessage.java      © ResponseMessage.java
 1 // Ariane Correa
 2 // ajcorrea
 3
 4 package org.example;
 5
 6 // Code Reference: EchoServerTCP.java from Coulouris text
 7
 8 import com.google.gson.JsonObject;
 9 import com.google.gson.JsonParser;
10
11 import java.io.BufferedReader;
12 import java.io.IOException;
13 import java.io.OutputStreamWriter;
14 import java.io.PrintWriter;
15 import java.net.ServerSocket;
16 import java.net.Socket;
17 import java.security.NoSuchAlgorithmException;
18 import java.sql.Timestamp;
19 import java.util.Scanner;
20
21 public class EchoServerTCP {
22
23     public static void main(String args[]) {
24
25         Socket clientSocket = null;
26
27         System.out.println("Blockchain server running");
28         try {
29             Scanner sc = new Scanner(System.in);
30             int serverPort = 7777; // Read user input
31             ServerSocket listenSocket = new ServerSocket(serverPort);
32
33             /*
34             * Block waiting for a new connection request from a client.
35             * When the request is received, "accept" it, and the rest
36             * the tcp protocol handshake will then take place, making
37             * the socket ready for reading and writing.
38             */
39             clientSocket = listenSocket.accept();
40             // If we get here, then we are now connected to a client.
41
42             // Set up "in" to read from the client socket
43             Scanner in;
44             in = new Scanner(clientSocket.getInputStream());
45
46             // Set up "out" to write to the client socket
47             PrintWriter out;
48             out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
49         }
```

```

50
51         /*
52          * Forever,
53          *   read a line from the socket
54          *   print it to the console
55          *   echo it (i.e. write it) back to the client
56         */
57
58         BlockChain chain = new BlockChain();
59         int option = 0;
60
61         Block genesis = new Block(index: 0, chain.getTime(), data: "Genesis", difficulty: 2);
62         genesis.setPreviousHash("");
63         genesis.proofOfWork();
64
65         chain.computeHashesPerSecond();
66         chain.addBlock(genesis);
67
68         System.out.println("We have a visitor");
69         while (true) {
70             JSONObject clientReply = new JsonParser().parse(in.nextLine()).getAsJsonObject();
71             option = clientReply.get("op").getAsInt();
72             switch (option) {
73                 // Status of blockchain
74                 case 0: {
75                     ResponseMessage responseMessage = new ResponseMessage(option,
76                         chain.getChainSize(), chain.getLatestBlock().getDifficulty(),
77                         chain.getTotalDifficulty(), chain.getHashesPerSecond(), chain.getTotalExpectedHashes(),
78                         chain.getLatestBlock().getNonce(), chain.getChainHash());
79                     System.out.println("Response : " + responseMessage.getResponseJson());
80                     out.println(responseMessage.getResponseJson());
81                     out.flush();
82                     break;
83                 }
84
85                 // Adding block to blockchain
86                 case 1: {
87                     System.out.println("Adding a block");
88                     int difficulty = clientReply.get("difficulty").getAsInt();
89                     String transaction = clientReply.get("transaction").getAsString();
90                     Timestamp start = chain.getTime();
91                     Block newBlock = new Block(chain.getChainSize(), chain.getTime(), transaction, difficulty);
92                     newBlock.setPreviousHash(chain.getChainHash());
93                     newBlock.proofOfWork();
94                     chain.addBlock(newBlock);
95                     Timestamp end = chain.getTime();
96                     ResponseMessage responseMessage = new ResponseMessage(option, response: "Total execution " +
97                         "time to add this block was " + (end.getTime() - start.getTime()) + " milliseconds");
98                     System.out.println("Setting response to : " + responseMessage.getResponseJson());
99                     out.println(responseMessage.getResponseJson());
100                    out.flush();
101                }
102            }

```

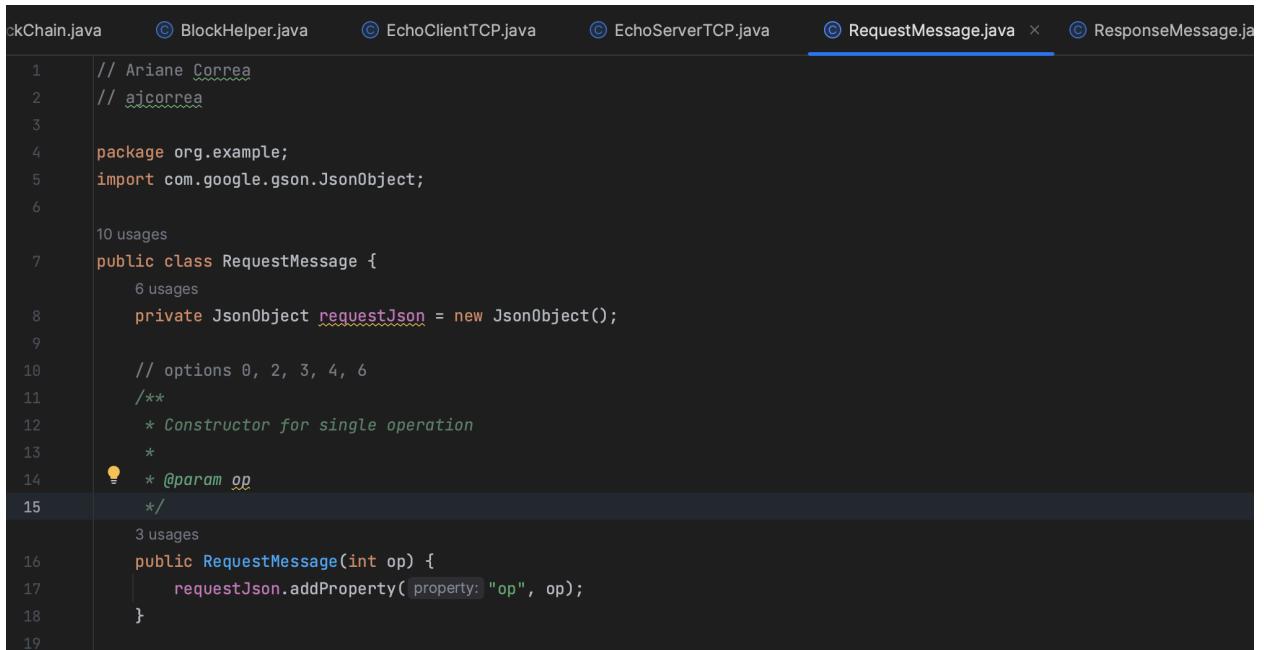
```

103     // Verify blockchain
104
105     case 2: {
106         System.out.println("Verifying entire chain");
107         Timestamp start = chain.getTime();
108         System.out.println("Chain verification: " + chain.isChainValid());
109         Timestamp end = chain.getTime();
110         ResponseMessage responseMessage = new ResponseMessage(option, response: "Total execution " +
111             "time to verify the chain was " + (end.getTime() - start.getTime()) + " milliseconds");
112         System.out.println("Setting response to: " + responseMessage.getResponseJson());
113         out.println(responseMessage.getResponseJson());
114         out.flush();
115         break;
116     }
117
118     // View blockchain
119     case 3: {
120         System.out.println("View the Blockchain");
121         ResponseMessage responseMessage = new ResponseMessage(option, chain.toString());
122         System.out.println("Setting response to: " + responseMessage.getResponseJson());
123         out.println(responseMessage.getResponseJson());
124         out.flush();
125         break;
126     }
127
128     // Corrupt blockchain
129     case 4: {
130         System.out.println("Corrupt the Blockchain");
131         int index = clientReply.get("index").getAsInt();
132         String transaction = clientReply.get("transaction").getAsString();
133         chain.getBlock(index).setData(transaction);
134         ResponseMessage responseMessage = new ResponseMessage(option, response: "Block " + index +
135             " now holds " + transaction);
136         System.out.println("Setting response to: " + responseMessage.getResponseJson());
137         out.println(responseMessage.getResponseJson());
138         out.flush();
139         break;
140     }
141
142     // Repair blockchain
143     case 5: {
144         System.out.println("Repairing the entire chain");
145         Timestamp start = chain.getTime();
146         chain.repairChain();
147         Timestamp end = chain.getTime();
148         ResponseMessage responseMessage = new ResponseMessage(option, response: "Total execution " +
149             "time required to repair the chain was " + (end.getTime() - start.getTime()) + " milliseconds");
150         System.out.println("Setting response to: " + responseMessage.getResponseJson());
151         out.println(responseMessage.getResponseJson());
152         out.flush();
153         break;
154     }
155
156
157     // Exit the program
158     case 6:
159         System.exit(status: 0);
160     }
161 }
```

```

162         // Handle exceptions
163     } catch (IOException e) {
164         System.out.println("IO Exception:" + e.getMessage());
165         // If quitting (typically by you sending quit signal) clean up sockets
166     } catch (NoSuchAlgorithmException e) {
167         throw new RuntimeException(e);
168     } finally {
169         try {
170             if (clientSocket != null) {
171                 clientSocket.close();
172             }
173         } catch (IOException e) {
174             // Ignore exception on close
175         }
176     }
177 }
178 }
179 }
```

## Code for RequestMessage.java



```

ckChain.java      ⚡ BlockHelper.java      ⚡ EchoClientTCP.java      ⚡ EchoServerTCP.java      ⚡ RequestMessage.java ×      ⚡ ResponseMessage.java
1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5 import com.google.gson.JsonObject;
6
7 10 usages
8 public class RequestMessage {
9     6 usages
10    private JsonObject requestJson = new JsonObject();
11
12    // options 0, 2, 3, 4, 6
13    /**
14     * Constructor for single operation
15     *
16     * @param op
17     */
18    3 usages
19    public RequestMessage(int op) {
20        requestJson.addProperty("property", "op", op);
21    }
22 }
```

```

20     // options 1 , 4
21     /**
22      * Constructor to send additional property and transaction details
23      *
24      * @param op
25      * @param property
26      * @param transaction
27      */
2 usages
28     public RequestMessage(int op, int property, String transaction) {
29         requestJson.addProperty( property: "op", op);
30         if (op == 1)
31             requestJson.addProperty( property: "difficulty", property);
32         if (op == 4)
33             requestJson.addProperty( property: "index", property);
34         requestJson.addProperty( property: "transaction", transaction);
35     }
36
37     /**
38      * Getter method for requestJson
39      *
40      * @return
41      */
4 usages
42     public JsonObject getRequestJson() {
43         return requestJson;
44     }
45 }
```

## Code for ResponseMessage.java



```

1 // Ariane Correa
2 // ajcorrea
3
4 package org.example;
5
6 import com.google.gson.JsonObject;
7 import java.math.BigInteger;
8
9 12 usages
10 public class ResponseMessage {
11     11 usages
12         private JsonObject responseJson = new JsonObject();
```

```

12 // option 0
13 /**
14 * Constructor for ResponseMessage option 0 with all parameters
15 *
16 * @param option
17 * @param chainSize
18 * @param difficulty
19 * @param totalDifficulty
20 * @param hashesPerSecond
21 * @param totalExpectedHashes
22 * @param nonce
23 * @param chainHash
24 */
25 usage
26 public ResponseMessage(int option, int chainSize, int difficulty, int totalDifficulty,
27                         int hashesPerSecond, double totalExpectedHashes, BigInteger nonce, String chainHash) {
28     responseJson.addProperty( property: "selection", option);
29     responseJson.addProperty( property: "size", chainSize);
30     responseJson.addProperty( property: "chainHash", chainHash);
31     responseJson.addProperty( property: "totalHashes", totalExpectedHashes);
32     responseJson.addProperty( property: "totalDiff", totalDifficulty);
33     responseJson.addProperty( property: "recentNonce", nonce);
34     responseJson.addProperty( property: "diff", difficulty);
35     responseJson.addProperty( property: "hps", hashesPerSecond);
36 }
37
38 // option 1
39 /**
40 * Constructor for ResponseMessage with selection message and response
41 *
42 * @param selection
43 * @param response
44 */
45 usages
46 public ResponseMessage(int selection , String response) {
47     responseJson.addProperty( property: "selection", selection);
48     responseJson.addProperty( property: "response", response);
49 }
50
51 /**
52 * getter method for ResponseJson
53 *
54 * @return
55 */
56 usages
57 public JsonObject getResponseJson() {
58     return responseJson;
59 }
```