Task 1 Server Source Code

```java
package blockchaintask;

import org.json.JSONObject;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.sql.Timestamp;
import java.util.HashMap;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.Scanner;

/**
 * class BlockChainServer
 * The server class that interacts with the blockchain to perform functions on the blockchain provided by the client
 */
public class BlockChainServer {

    /**
     * function chain
     * The chain function is used to perform various operations on the blockchain
     * @param in the client input
     * @param out output to client
     * @param blockChain the blockchain
     */
    public static void chain(Scanner in, PrintWriter out, BlockChain blockChain) {
        //If the size of the blocks is 0 then add the genesis block
        if(blockChain.blocks.size() == 0) {
            //creating a block and its previous hash
            Block block = new Block(0, blockChain.getTime(), "Genesis", 2);
            block.setPreviousHash("");
            try {
                //doing the proof of work of the block
                block.proofOfWork();
                //adding the block in the array list
                blockChain.blocks.add(block);
                blockChain.computeHashesPerSecond(); //computing the hashes per second
                blockChain.setChainHash(block.calculateHash()); //setting the chain hash
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
        }
        //Json object to get the data from the client
        JSONObject json = new JSONObject(in.nextLine());
        //Checking the choice sent by the client and performing functions based on that
        int choice = Integer.parseInt(json.getString("choice"));
        JSONObject jsonResponse;
        Map<String, String> responseMap = new HashMap<>();
        switch(choice)
        {   //Sending the data of the blockchain. The chain size, difficulty of the latest block, total difficulty
            //the hashes per second computed above, total expected hashes by computing expected hashes of each block
            //Nonce of the latest block and the chainHash i.e the hash of the latest block
            case 0: responseMap = new HashMap<>();
                responseMap.put("chainSize", String.valueOf(blockChain.getChainSize()));
                responseMap.put("difficulty", String.valueOf(blockChain.getLatestBlock().getDifficulty()));
                responseMap.put("totalDifficulty",
String.valueOf(blockChain.getTotalDifficulty()));
                responseMap.put("hashesPerSecond", String.valueOf(blockChain.getHashesPerSecond()));
                responseMap.put("totalExpectedHashes", String.valueOf(blockChain.getTotalExpectedHashes()));
                responseMap.put("nonce", String.valueOf(blockChain.getLatestBlock().getNonce()));
                responseMap.put("chainHash", blockChain.getChainHash());
                break;
            //Adding a transaction to the block, getting the difficulty and the transaction data from the client
            case 1:
                int difficulty = Integer.parseInt(json.getString("difficulty"));
                String data = json.getString("data");
                Timestamp startTime = blockChain.getTime();
                //creating the new block
                Block block1 = new Block(blockChain.getChainSize(), blockChain.getTime(), data, difficulty);
                block1.setPreviousHash(blockChain.getChainHash()); //setting the previous hash of the block
                try {
                    block1.proofOfWork(); //proof of work for the block
                    blockChain.addBlock(block1); //adding the block to the chain
                } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
                }
                Timestamp endTime = blockChain.getTime();
                long totalTime = endTime.getTime() - startTime.getTime();
```

```java
        responseMap = new HashMap<>();
        //Sending back the time taken to add the block
        responseMap.put("time", String.valueOf(totalTime));
        break;
        //verifying the chain
      case 2: boolean isValid = false;
        Timestamp startTimeValid = blockChain.getTime();
        try {
          isValid = blockChain.isChainValid();
        } catch (NoSuchAlgorithmException e) {
          e.printStackTrace();
        }
        Timestamp endTimeValid = blockChain.getTime();
        responseMap = new HashMap<>();
        //sending true if the chain is valid
        if(blockChain.getErrorMessage().equals("")) {
          responseMap.put("isValid", String.valueOf(isValid));
          responseMap.put("time", String.valueOf(endTimeValid.getTime() - startTimeValid.getTime()));
        }
        else //sending an error message in json object if the chain is not valid
        {
          responseMap.put("isValid", String.valueOf(isValid));
          responseMap.put("errorMessage", blockChain.getErrorMessage());
          responseMap.put("time", String.valueOf(endTimeValid.getTime() - startTimeValid.getTime()));
          blockChain.setErrorMessage("");
        }
```

```java
                break;
                //Sending the toString of the entire chain to the client
            case 3: responseMap = new HashMap<>();
                responseMap.put("blockchain", blockChain.toString());
                break;
                //Corrupting the data of the block if given by the client
            case 4: int id = Integer.parseInt(json.getString("id"));
                String corruptData = json.getString("data");
                blockChain.getBlock(id).setData(corruptData); //adding the corrupt data
                responseMap = new HashMap<>();
                //sending a message that the data is corrupted for the id
                responseMap.put("id", String.valueOf(id));
                responseMap.put("corruptData", corruptData);
                break;
                //Repairing the blockchain
            case 5: Timestamp startTimeRepair = blockChain.getTime();
                try {
                    blockChain.repairChain();
                } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
                }
                //sending the time taken to repair the blockchain
                Timestamp endTimeRepair = blockChain.getTime();
                responseMap = new HashMap<>();
                responseMap.put("time", String.valueOf(endTimeRepair.getTime() - startTimeRepair.getTime()));
                break;
```

```java
        }
        //The maps created during the choice, sending them in the jsonResponse to the client
        jsonResponse = new JSONObject(responseMap);
        out.println(jsonResponse);
        out.flush();
    }

    /**
     * function main
     * The function is used to connect with the client. The server socket listens to the client.
     * @param args
     */
    public static void main(String args[]) {
        Socket clientSocket = null;
        BlockChain blockChain = new BlockChain();
        try {
            int serverPort = 7777; // the server port we are using

            // Create a new server socket
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while (true) {
                clientSocket = listenSocket.accept();
                //"in" to read from the client socket
                Scanner in = new Scanner(clientSocket.getInputStream());
                //"out" to write to the client socket
                PrintWriter out;
                out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
                //verifying the message and returning the computation result
                chain(in, out, blockChain);
            }

            // Handle exceptions
        } catch (IOException e) { //Checking for input output exceptions
        } catch (NoSuchElementException e) {
        } finally { //If socket is not null and request is done, close the socket
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }
}

```