Practical No.9

Name: Bhairavi Narendra Rewatkar

Roll No.: DMET1221006

Subject: Blockchain Technology Laboratory

Title: P2P Network

Aim: Write a program to implement a basic P2P network for nodes to communicate.

Source Code:

Node.java

```
import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.List;
class Node {
    private String nodeName;
    private Blockchain blockchain;
    private List<Node> connections;
    private ServerSocket serverSocket;
    public Node(String nodeName, int difficulty) {
        this.nodeName = nodeName;
        this.blockchain = new Blockchain(difficulty);
        this.connections = new ArrayList<>();
        try {
            // Initialize the server socket for this node to accept incoming connections
            serverSocket = new ServerSocket(0); // Bind to an available port
            System.out.println(nodeName + " is running on port " + serverSocket.getLocalPort());
            startServer();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Method to start a server to handle incoming connections
    public void startServer() {
```

```
        new Thread(() -> {
            while (true) {
                try {
                    Socket socket = serverSocket.accept(); // Accept incoming connections
                    new Thread(() -> handleIncomingConnection(socket)).start(); // Handle each connection
on a new thread
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
    // Method to connect this node to another node using a host and port
    public void connectToNode(String host, int port) {
        try {
            Socket socket = new Socket(host, port);
            connections.add(this); // Add this node to the connection list
            System.out.println(nodeName + " connected to " + host + ":" + port);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Method to handle incoming connections and process messages
    public void handleIncomingConnection(Socket socket) {
        try {
            BufferedReader inputStream = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String message;
            while ((message = inputStream.readLine()) != null) {
                processMessage(message);
            }
        } catch (IOException e) {
            e.printStackTrace();
```

```
    }
}
// Method to send a message to another node
public void sendMessage(Node node, String message) {
    try {
        Socket socket = new Socket(node.getHost(), node.getPort());
        PrintWriter outputStream = new PrintWriter(socket.getOutputStream(), true);
        outputStream.println(message);
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
// Method to add a transaction and broadcast it to other connected nodes
public void addTransaction(Transaction transaction) {
    blockchain.addTransaction(transaction);
    for (Node connectedNode : connections) {
        sendMessage(connectedNode, "NEW_TRANSACTION:" + transaction.toString());
    }
}
// Method to mine a block and broadcast it to other connected nodes
public void mineBlock() {
    blockchain.mineBlock();
    for (Node connectedNode : connections) {
        sendMessage(connectedNode, "NEW_BLOCK:" + blockchain.getLastBlock().toString());
    }
}
// Method to process incoming messages
public void processMessage(String message) {
    // Handle new transactions or blocks
    if (message.startsWith("NEW_TRANSACTION:")) {
        String transactionData = message.substring("NEW_TRANSACTION:".length());
```

```
            System.out.println(nodeName + " received transaction: " + transactionData);
        } else if (message.startsWith("NEW_BLOCK:")) {
            String blockData = message.substring("NEW_BLOCK:".length());
            System.out.println(nodeName + " received block: " + blockData);
        }
    }
    // Placeholder methods for node address
    public String getHost() {
        return "localhost"; // Placeholder: replace with actual host info if needed
    }
    public int getPort() {
        return serverSocket.getLocalPort();
    }
}
```

Transaction.java

```
class Transaction {
    private String sender;
    private String receiver;
    private double amount;

    public Transaction(String sender, String receiver, double amount) {
        this.sender = sender;
        this.receiver = receiver;
        this.amount = amount;
    }
    @Override
    public String toString() {
        return "Transaction{" +
                "sender='" + sender + '\'' +
                ", receiver='" + receiver + '\'' +
                ", amount=" + amount +
                '}';
```

```
    }
}
```

Block.java

```
import java.util.List;
class Block {
    private String previousHash;
    private String hash;
    private List<Transaction> transactions;
    private int nonce;
    private int difficulty;
    public Block(String previousHash, List<Transaction> transactions, int difficulty) {
        this.previousHash = previousHash;
        this.transactions = transactions;
        this.difficulty = difficulty;
        this.hash = ""; // Initialize hash with an empty value
        this.hash = mineBlock(); // Generate the valid hash
    }
    // Method to compute hash by applying Proof-of-Work
    public String mineBlock() {
        String target = new String(new char[difficulty]).replace('\0', '0'); // Create difficulty target
        String calculatedHash = ""; // Temporary variable for hash calculation
        while (!calculatedHash.startsWith(target)) {
            nonce++;
            calculatedHash = HashUtil.applySHA256(previousHash + transactions.toString() + nonce);
        }
        return calculatedHash;
    }

    public String getHash() {
        return this.hash;
    }
```

```
    @Override
    public String toString() {
        return "Block{" +
                "previousHash='" + previousHash + '\'' +
                ", hash='" + hash + '\'' +
                ", transactions=" + transactions +
                ", nonce=" + nonce +
                '}';
    }
}
```

Blockchain.java

```
import java.util.ArrayList;
import java.util.List;

class Blockchain {
    private List<Block> chain;
    private List<Transaction> transactionPool;
    private int difficulty;

    public Blockchain(int difficulty) {
        this.difficulty = difficulty;
        this.chain = new ArrayList<>();
        this.transactionPool = new ArrayList<>();
        chain.add(createGenesisBlock());
    }

    // Method to create the Genesis Block (first block in the blockchain)
    private Block createGenesisBlock() {
        return new Block("0", new ArrayList<>(), difficulty);
    }

    // Method to add a transaction to the transaction pool
```

```
    public void addTransaction(Transaction transaction) {
        transactionPool.add(transaction);
    }
    // Method to mine a new block with all pending transactions
    public void mineBlock() {
        Block newBlock = new Block(chain.get(chain.size() - 1).getHash(), new
ArrayList<>(transactionPool), difficulty);
        chain.add(newBlock);
        transactionPool.clear(); // Clear the transaction pool after mining
        System.out.println("Block mined: " + newBlock);
    }
    // Method to get the latest block in the chain
    public Block getLastBlock() {
        return chain.get(chain.size() - 1);
    }
}
```

HashUtil.java

```
import java.security.MessageDigest;
class HashUtil {
    public static String applySHA256(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(input.getBytes("UTF-8"));
            StringBuilder hexString = new StringBuilder();
            for (byte b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
```

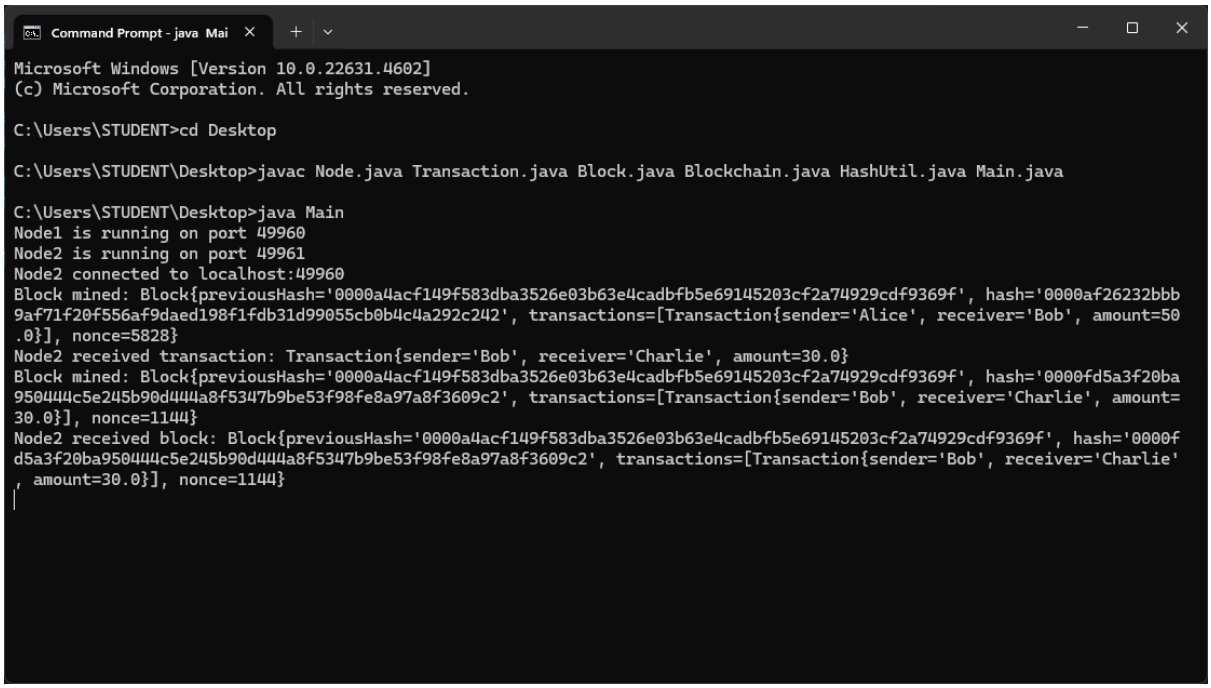
```
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Node firstNode = new Node("Node1", 4);
        Node secondNode = new Node("Node2", 4);
        // Connect second node to the first node
        secondNode.connectToNode("localhost", firstNode.getPort());
        // Perform transactions and mining
        firstNode.addTransaction(new Transaction("Alice", "Bob", 50));
        firstNode.mineBlock();

        secondNode.addTransaction(new Transaction("Bob", "Charlie", 30));
        secondNode.mineBlock();
    }
}
```

Output:



```
Command Prompt - java Mai
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\STUDENT>cd Desktop

C:\Users\STUDENT\Desktop>javac Node.java Transaction.java Block.java Blockchain.java HashUtil.java Main.java

C:\Users\STUDENT\Desktop>java Main
Node1 is running on port 49960
Node2 is running on port 49961
Node2 connected to localhost:49960
Block mined: Block{previousHash='0000a4acf149f583dba3526e03b63e4cadbfb5e69145203cf2a74929cdf9369f', hash='0000af26232bbb
9af71f20f556af9daed198f1fdb31d99055cb0b4c4a292c242', transactions=[Transaction{sender='Alice', receiver='Bob', amount=50
.0}], nonce=5828}
Node2 received transaction: Transaction{sender='Bob', receiver='Charlie', amount=30.0}
Block mined: Block{previousHash='0000a4acf149f583dba3526e03b63e4cadbfb5e69145203cf2a74929cdf9369f', hash='0000fd5a3f20ba
950444c5e245b90d444a8f5347b9be53f98fe8a97a8f3609c2', transactions=[Transaction{sender='Bob', receiver='Charlie', amount=
30.0}], nonce=1144}
Node2 received block: Block{previousHash='0000a4acf149f583dba3526e03b63e4cadbfb5e69145203cf2a74929cdf9369f', hash='0000f
d5a3f20ba950444c5e245b90d444a8f5347b9be53f98fe8a97a8f3609c2', transactions=[Transaction{sender='Bob', receiver='Charlie'
, amount=30.0}], nonce=1144}
```