Blockchain Lab Project File:

Rahul Sahani_22bcon1137

Section P (UPGRAD)

B. Tech (CSE)

#1 Write a program that takes a string input and generates a SHA-256 hash of the input. Demonstrate the usage of the function by generating hashes for different input strings.

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
public class Source {
   private static final Scanner scan = new Scanner(System.in);
   public static void main(String[] args) throws NoSuchAlgorithmException {
      System.out.print("Enter a Message to want to Hash: ");
      String message = scan.nextLine();
      String hashedMessage = hashCreator(message);
      System.out.println("Original Message: " + message);
      System.out.println("Hashed Message: " + hashedMessage); }
   public static String hashCreator(String input) throws NoSuchAlgorithmException {
      MessageDigest md = MessageDigest.getInstance("SHA-256");
      md.update(input.getBytes());
      byte[] digest = md.digest();
      StringBuilder hexMessage = new StringBuilder();
      for (byte b : digest) {
         hexMessage.append(String.format("%02x", b)); }
      return hexMessage.toString(); }
```

#2 Write a program that simulates a network of nodes reaching a consensus on the validity of a new block.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
public class Source {
   public static void main(String[] args) {
```

```
List<Node> nodes = new ArrayList<>();
        nodes.add(new Node("Node 1"));
        // Repeat the above step for 4 more nodes.
        nodes.add(new Node("Node 2"));
        nodes.add(new Node("Node 3"));
        nodes.add(new Node("Node 4"));
        nodes.add(new Node("Node 5"));
        Scanner scanner = new Scanner(System.in);
        int validityRate = scanner.nextInt();
        Block newBlock = generateNewBlock();
        for (Node node : nodes) {
            node.receiveBlock(newBlock);}
        boolean isBlockAccepted = validateBlock(nodes, newBlock, validityRate);
        if (isBlockAccepted) {
            System.out.println("Block accepted by the network.");
        } else {
            System.out.println("Block rejected by the network."); }
        scanner.close(); }
    private static Block generateNewBlock() {
        int index = 1;
        String data = "Dummy Data";
        String previousHash = "Dummy Previous Hash";
        return new Block(index, data, previousHash); }
    private static boolean validateBlock(List<Node> nodes, Block newBlock, int validityRate) {
        int majorityThreshold = nodes.size() / 2 + 1;
        int validNodeCount = 0;
        Random random = new Random();
        for (Node node : nodes) {
            if (node.validateBlock(newBlock, validityRate, random)) {
                validNodeCount++; }}
        return validNodeCount >= majorityThreshold;}}
class Block {
    private int index;
    private String data;
    private String previousHash;
    public Block(int index, String data, String previousHash) {
        this.index = index;
        this.data = data;
        this.previousHash = previousHash;}
    public int getIndex() {
        return index;}
    public String getData() {
        return data;}
    public String getPreviousHash() {
        return previousHash;}}
```

```
class Node {
   private String name;
   public Node(String name) {
      this.name = name; }
   public void receiveBlock(Block block) {
      // System.out.println(name + " received a new block: " + block); }
   public boolean validateBlock(Block block, int validityRate, Random random) {
      int randomNumber = random.nextInt(100);
      boolean isValid = randomNumber < validityRate;
      return isValid; }}
```

#3 Write a program to simulate a basic blockchain system where the user can input data for each block, including the previous hash. The program then generates a hash for each block and organises the blocks in the blockchain based on their timestamps in ascending order.

```
import java.util.*;
public class Source {
   public static void main(String[] args) {
      List<Block> blockchain = new ArrayList<>();
      Scanner scanner = new Scanner(System.in);
      int index = 1;
      long[] timestamps = {
            1624388502000L,
            1624388501000L,
            1624388503000L,
            1624388499000L };
      int timestampIndex = 0;
      while (true) {
         String data = scanner.nextLine();
         if (data.equals("q")) break;
         String previousHash = scanner.nextLine();
         blockchain.add(new Block(index, timestamps[timestampIndex], data, previousHash));
         index++;
         timestampIndex++; }
Collections.sort(blockchain, Comparator.comparing(Block::getTimestamp));
      for (Block block : blockchain) {
         System.out.println("Index: " + block.getIndex()); } } }
class Block {
      private int index;
      private long timestamp;
      private String data;
      private String previousHash;
      private String hash;
```

```
    public Block(int index, long timestamp, String data, String previousHash) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.previousHash = previousHash;
        this.hash = calculateHash(); }
    public int getIndex() {
        return index; }
    public long getTimestamp() {
        return timestamp; }
    public String getData() {
        return data; }
    public String getPreviousHash() {
        return previousHash; }
    public String getHash() {
        return hash; }
    private String calculateHash() {
        return data + previousHash + "hashed"; }}
```

#4. Write a Java program to implement a Block class for a blockchain. The Block should have attributes like index, timestamp, data, previous hash, and hash. Include a method to generate the hash value based on the block's properties.

```
pragma solidity >=0.6.0 <0.7.0;

contract MyContract{

uint public data;
   uint public data1;
   uint public data2;
   uint public data3;
   uint public data4;
   bytes32 public hash;
   bytes32 public hash256;
   bytes32 public block_hash;
   bool public b1;
   bool public b2;

function pub() public {
    data1 = addmod (11,20,5);
    data2 = mulmod (12,4,5);
}

function chkbitwise() public {
```

```
    b1 = true && false;
    b2 = true || false;
}
function chkhash() public {
    hash = keccak256("Blockchain");
    hash256 = sha256("Block 2");
    block_hash = blockhash(0);
    data = block.difficulty;
    data3 = block.number;
    data4 = block.gaslimit;
}
}
```

#5. Write a program to simulate a blockchain transaction verifier. It should verify the integrity of a transaction by checking its digital signature, and it should also ensure that the sender has sufficient funds to perform the transaction.

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

contract Money{

    address alice = 0x5c6B0f7Bf3E7ce046039Bd8FABdfD3f9F5021678;
    //balance --> check the balance of the address
    //transfer --> used to send/transfer to the respective address

    function getMoney() public payable{}

    //function TransferMoney(uint _amount) public {
        //payable(alice).transfer(_amount);
    //}

    //fallback() external payable{}

    function TransferMoney() public {
        payable(alice).transfer(address(this).balance);
    }
}
```

#6 Implement a Java program to validate the integrity of a blockchain. The function should take an array of blocks as input and verify the correctness of each block, including the validity of transactions, the consistency of hashes, and the linkage between blocks using the previous hash. Demonstrate the usage of the function by validating a given blockchain.

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
class Block {
    private final int index;
    private final long timestamp;
    private final String data;
    private final String previousHash;
    private final String hash;
    public Block(int index, long timestamp, String data, String previousHash) {
        this.index = index;
        this.timestamp = timestamp;
        this.data = data;
        this.previousHash = previousHash;
        this.hash = calculateHash();
    }
    public int getIndex() {
        return index;
    }
    public long getTimestamp() {
        return timestamp;
    }
    public String getData() {
        return data;
    }
    public String getPreviousHash() {
        return previousHash;
    }
    public String getHash() {
        return hash;
    }
    public String calculateHash() {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            String dataToHash = index + timestamp + data + previousHash;
            byte[] hashBytes = digest.digest(dataToHash.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();
            for (byte hashByte : hashBytes) {
                String hex = String.format("%02x", hashByte);
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
```

```
        return null;
    }
}
public class BlockchainValidity {
    public static boolean validateBlockchain(Block[] blockchain) {
        // Check the validity of each block in the blockchain
        for (int i = 1; i < blockchain.length; i++) {
            Block currentBlock = blockchain[i];
            Block previousBlock = blockchain[i - 1];
            // Verify the consistency of hashes
            if (!currentBlock.getHash().equals(currentBlock.calculateHash())) {
                return false;
            }
            // Verify the linkage between blocks
            if (!currentBlock.getPreviousHash().equals(previousBlock.getHash())) {
                return false;
            }
        }
        // All blocks are valid
        return true;
    }
    public static void main(String[] args) {
        // Create a sample blockchain
        Block[] blockchain = new Block[3];
        blockchain[0] = new Block(0, System.currentTimeMillis(), "Genesis Block", "0");
        blockchain[1] = new Block(1, System.currentTimeMillis(), "Block 1",
blockchain[0].getHash());
        blockchain[2] = new Block(2, System.currentTimeMillis(), "Block 2",
blockchain[1].getHash());
        // Validate the blockchain
        boolean isValid = validateBlockchain(blockchain);
        // Print the validation result
        if (isValid) {
            System.out.println("Blockchain is valid.");
        } else {
            System.out.println("Blockchain is not valid.");
        }
    }
}
```

#7 Write a program to implement a patient record management system for a medical facility. The system involves various classes to model patients, doctors, medical records, checkup results, and prescriptions.

This system acts as a digital bridge connecting patients, doctors, and pharmacists to ensure comprehensive healthcare management while incorporating blockchain technology for enhanced security and interoperability.

The system needs to allow doctors to perform checkups, record results, and diagnose a patient, while patients are added to the system with their information and checkup details. The code also needs to allow a chemist to access the patient's diagnosis and provide prescriptions.

```
Import java.util.ArrayList;
Import java.util.HashMap;
Import java.util.Map;

Class Patient {
   String name;
   Int age;
   String address;

   Patient(String name, int age, String address) {
      This.name = name;
      This.age = age;
      This.address = address;
   }
}

Class Doctor {
   String name;
   String specialization;

   Doctor(String name, String specialization) {
      This.name = name;
      This.specialization = specialization;
   }

   MedicalRecord performCheckup(Patient patient, String diagnosis) {
      MedicalRecord medicalRecord = new MedicalRecord(patient, this, diagnosis);
      MedicalRecordSystem.addMedicalRecord(medicalRecord);
      Return medicalRecord;
   }
}

Class MedicalRecord {
   Patient patient;
   Doctor doctor;
   String diagnosis;
```

```
    CheckupResult checkupResult;
    Prescription prescription;

    MedicalRecord(Patient patient, Doctor doctor, String diagnosis) {
        This.patient = patient;
        This.doctor = doctor;
        This.diagnosis = diagnosis;
    }

    Void recordCheckupResult(CheckupResult result) {
        This.checkupResult = result;
    }

    Void providePrescription(Prescription prescription) {
        This.prescription = prescription;
    }
}

Class CheckupResult {
    String result;

    CheckupResult(String result) {
        This.result = result;
    }
}

Class Prescription {
    String medication;

    Prescription(String medication) {
        This.medication = medication;
    }
}

Class MedicalRecordSystem {
    Static ArrayList<MedicalRecord> medicalRecords = new ArrayList<>();
    Static Map<Patient, MedicalRecord> prescriptions = new HashMap<>();

    Static void addMedicalRecord(MedicalRecord medicalRecord) {
        medicalRecords.add(medicalRecord);
    }

    Static MedicalRecord getMedicalRecord(Patient patient) {
        For (MedicalRecord record : medicalRecords) {
            If (record.patient.equals(patient)) {
```

```
            Return record;
          }
        }
        Return null;
    }

    Static void providePrescription(Patient patient, Prescription prescription) {
        MedicalRecord medicalRecord = prescriptions.get(patient);
        If (medicalRecord != null) {
            medicalRecord.providePrescription(prescription);
        }
    }
}

Class Chemist {
    Void providePrescription(Patient patient, Prescription prescription) {
        MedicalRecordSystem.providePrescription(patient, prescription);
    }
}

Public class HealthcareSystem {
    Public static void main(String[] args) {
        // Example Usage
        Patient patient1 = new Patient("John Doe", 30, "123 Main St");
        Doctor doctor1 = new Doctor("Dr. Smith", "Cardiologist");

        // Doctor performs checkup and records diagnosis
        MedicalRecord medicalRecord = doctor1.performCheckup(patient1, "High blood
pressure");

        // Chemist accesses patient's diagnosis and provides prescription
        Chemist chemist = new Chemist();
        Prescription prescription = new Prescription("Blood pressure medication");
        Chemist.providePrescription(patient1, prescription);
    }
}
```