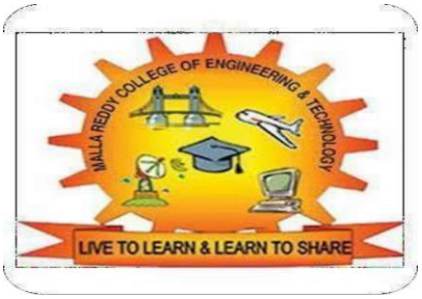
BLOCKCHAIN TECHNOLOGY LAB [R20A0591]

LABORATORY MANUAL

B. TECH IT (IV YEAR–I SEM)

R20 REGULATION(2024-25)



PREPARED BY
M.VAZRALU
G.LAVARAJU

DEPARTMENT INFORMATION TECHNOLOGY

MALLAREDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution– UGC, Govt. of India)
Recognized under2(f)and12(B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA &NAAC– 'A'Grade-ISO9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad –500100, Telangana,India

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION

- To achieve high quality in technical education that provides the skills and attitude to adapt to the global needs of the Information Technology sector, through academic and research excellence.

MISSION

- To equip the students with the cognizance for problem solving and to improve the teaching pedagogy by using innovative techniques.
- To strengthen the knowledge base of the faculty and students with motivation towards possession of effective academic skills and relevant research experience
- To promote the necessary moral and ethical values among the engineers, for the betterment of the society

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: PROFESSIONALISM & CITIZENSHIP To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.
PEO2: TECHNICAL ACCOMPLISHMENTS To provide knowledge-based services to satisfy the needs of society and the industry by providing hands on experience in various technologies in core field.
PEO3: INVENTION, INNOVATION AND CREATIVITY To make the students to design, experiment, analyze, interpret in the core field with the help of other multi-disciplinary concepts wherever applicable.
PEO4: PROFESSIONAL DEVELOPMENT To educate the students to disseminate research findings with good soft skills and become a successful entrepreneur.
PEO5: HUMAN RESOURCE DEVELOPMENT To graduate the students in building national capabilities in technology, education and research.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Information Technology, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

Applications of Computing Domain & Research: Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF INFORMATION TECHNOLOGY

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at-least 5 minutes before (to the starting time), those who come after 5minutes will not be allowed into the lab.

2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.

3. Student should enter into the laboratory with:

   a) Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.

   b) Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.

   c) Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.

5. Execute your task in the laboratory, and record the results/output in the lab observation notebook, and get certified by the concerned faculty.

6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.

7. Computerlabsareestablishedwithsophisticatedandhighendbrandedsystems, which should be utilized properly.

8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.

9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.

10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

HEAD OF THE DEPARTMENT **PRINCIPAL**

MALLAREDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

IV Year B.Tech. IT I Sem **L/T/P/C**

-/-/3/1.5

(R20A0591)BLOCKCHAIN TECHNOLOGY LAB

COURSE OBJECTIVES

This course will enable the students:

1. Understanding Block chain Fundamentals and creating basic blocks.
2. Able to Develop Block chain Applications in a structured manner
3. Ability to create own crypto currency and get familiarity with future currencies.
4. Able to Evaluate and Analyze Block chain Systems

LIST OF EXPERIMENTS

Week 1: Creating Merkle tree

Week 2: Creation of Block

Week 3: Block chain Implementation Programming code

Week 4: Creating ERC20 token

Week 5: Java code to implement blockchain in Merkle Trees

Week 6: Java Code to implement Mining using block chain

Week 7: Java Code to implement peer-to-peer using block chain

Week 8: Creating a Crypto-currency Wallet

COURSE OUTCOMES

1. Knowledge of Blockchain Concepts and creating basic blocks.
2. Proficiency in Blockchain Development.
3. Ability to Design and Implement Blockchain Applications.
4. Evaluation and Analysis of Blockchain Systems.
5. Knowledge of crypto currency and creating a basic form of it.

TABLE OF CONTENTS

WEEK: 1

AIM: Creating Merkle tree

Merkle Tree

Merkle tree is a tree data structure with leaf nodes and non leaf nodes. It also known as Hash tree. The reason behind it is it only stores the hashes in its nodes instead of data. In its leaf nodes, it will store the hash of the data. Non leaf nodes contain the hash of its children.

Bit coin's merkle-tree implementation works the following way:

1. split the transactions in the block up into pairs
2. byte-swap the txids
3. concatenate the txids
4. double hash the concatenated pairs

SOURCE CODE:

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
public class MerkleTree {
private List<String> transactions;
private List<String> merkleTree;
public MerkleTree(List<String> transactions) {
this.transactions = transactions;
this.merkleTree = buildMerkleTree(transactions);
}
private String calculateHash(String data) {
try {
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hashBytes = digest.digest(data.getBytes(StandardCharsets.UTF_8));
StringBuilder hexString = new StringBuilder();
for (byte hashByte : hashBytes) {
        String hex = Integer.toHexString(0xff &hashByte);
if (hex.length() == 1) {
```

```
hexString.append('0');
}
hexString.append(hex);
}
return hexString.toString();
}
catch (NoSuchAlgorithmException e) {
e.printStackTrace();
}
return null;
}
private List<String> buildMerkleTree(List<String> transactions) {
List<String> merkleTree = new ArrayList<>(transactions);
int levelOffset = 0;
for (int levelSize = transactions.size(); levelSize> 1; levelSize = (levelSize + 1) / 2) {
for (int left = 0; left <levelSize; left += 2) {
int right = Math.min(left + 1, levelSize - 1);
String leftHash = merkleTree.get(levelOffset + left);
String rightHash = merkleTree.get(levelOffset + right);
String parentHash = calculateHash(leftHash + rightHash);
merkleTree.add(parentHash);
}
levelOffset += levelSize;
}
return merkleTree;
}
public List<String>getMerkleTree() {
return merkleTree;
}
public static void main(String[] args) {
List<String> transactions = new ArrayList<>();
transactions.add("Transaction 1");
transactions.add("Transaction 2");
transactions.add("Transaction 3");
transactions.add("Transaction 4");
```

```
MerkleTree merkleTree = new MerkleTree(transactions);
List<String> tree = merkleTree.getMerkleTree();
for (String hash : tree) {
System.out.println(hash);
}}}
```

EXPECTED OUTPUT:

```
Transaction 1
Transaction 2
Transaction 3
Transaction 4
39704f929d837dc8bd8e86c70c4fb06cf740e7294f1036d030e92fe545f18275
64833afa7026409be938e6e21a643749233e5d418b906fe5b6f304e7a7636eef
0bc1c5cf4cc8f4915cdf888eca02682416c6be663d7706b9fb0933038ab9981a
```

OUTPUT:

WEEK : 2

AIM : Creation of Block

Blocks are data structures within the blockchain database, where transaction data in a crypto currency blockchain are permanently recorded. A block records some or all of the most recent transactions not yet validated by the network. Once the data are validated, the block is closed. Then, a new block is created for new transactions to be entered into and validated.

Blocks are created when miners or block validation successfully validate the encrypted information in the block header, which prompts the creation of a new block.

SOURCE CODE:

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Date;
public class Block {
private int index;
private long timestamp;
private String previousHash;
private String hash;
private String data;
private int nonce;
public Block(int index, String previousHash, String data) {
this.index = index;
this.timestamp = new Date().getTime();
this.previousHash = previousHash;
this.data = data;
this.nonce = 0;
this.hash = calculateHash();
}
public String calculateHash() {
try {
MessageDigest digest = MessageDigest.getInstance("SHA-256");
String input = index + timestamp + previousHash + data + nonce;
byte[] hashBytes = digest.digest(input.getBytes());
StringBuilder hexString = new StringBuilder();
```

```
for (byte hashByte : hashBytes) {
String hex = Integer.toHexString(0xff &hashByte);
if (hex.length() == 1) {
hexString.append('0');
}
hexString.append(hex);
}
return hexString.toString();
}
catch (NoSuchAlgorithmException e) {
e.printStackTrace();
}
return null;
}
public void mineBlock(int difficulty) {
String target = new String(new char[difficulty]).replace('\0', '0');
while (!hash.substring(0, difficulty).equals(target)) {
nonce++;
hash = calculateHash();
}
System.out.println("Block mined: " + hash);
}
public int getIndex() {
return index;
}
public long getTimestamp() {
return timestamp;
}
public String getPreviousHash() {
return previousHash;
}
public String getHash() {
return hash;
}
public String getData() {
```

```
return data;
}
public static void main(String args[]){
Block b=new
Block(1,"3a42c503953909637f78dd8c99b3b85ddde362415585afc11901bdefe8349102","hai");
b.calculateHash();
b.mineBlock(1);
b.getIndex();
b.getTimestamp();
b.getPreviousHash();
b.getHash();
b.getData();
}}
```

EXPECTED OUTPUT:

```
Block mined: 0afa2aa66eacfd6cc776c8cd7856e354d52303a699bed38560de49efebd9cce3
```

OUTPUT:

WEEK:3

AIM : Block chain Implementation Programming code

Block chain programming fundamentals:

In order to understand Blockchain deeply, the concept of a Digital Signature or a Hash is important. Digital Signature is basically a function that takes a string as input and returns a fixed-size alphanumeric string. The output string is known as the Digital Signature or the Hash of the input message. The important point is that the function via which we obtain the Digital Signature is "irreversible" in that given an input string, it can compute the Hash. However, given the Hash, it is virtually impossible to compute the input string. Further, it is also virtually impossible to find 2 values that have the same Hash.

Hash1=hash(input1)

Hash2=hash(input2)

It is easy to compute hash1 from input1 and hash2 from input2.

It is virtually impossible to compute input1 given the value of hash1. Similarly for input2 and hash2.

It is virtually impossible to find distinct input1 and input2 such that hash1 = hash2.

SOURCE CODE:

```
import java.util.ArrayList;
import java.util.List;
public class Blockchain {
private List<Block> chain;
private int difficulty;
public Blockchain(int difficulty) {
this.chain = new ArrayList<>();
this.difficulty = difficulty;
// Create the genesis block
createGenesisBlock();
}
private void createGenesisBlock() {
Block genesisBlock = new Block(0, "0", "Genesis Block");
genesisBlock.mineBlock(difficulty);
chain.add(genesisBlock);
}
```

```
public Block getLatestBlock() {
return chain.get(chain.size() - 1);
}
public void addBlock(Block newBlock) {
newBlock.mineBlock(difficulty);
chain.add(newBlock);
}
public boolean isChainValid() {
for (int i = 1; i <chain.size(); i++) {
Block currentBlock = chain.get(i);
Block previousBlock = chain.get(i - 1);
if (!currentBlock.getHash().equals(currentBlock.calculateHash())) {
System.out.println("Invalid hash for Block " + currentBlock.getIndex());
return false;
}
if (!previousBlock.getHash().equals(currentBlock.getPreviousHash())) {
System.out.println("Invalid previous hash for Block " + currentBlock.getIndex());
return false;
}}
return true;
}
public static void main(String[] args) {
Blockchain blockchain = new Blockchain(4);
Block block1 = new Block(1, blockchain.getLatestBlock().getHash(), "Data 1");
blockchain.addBlock(block1);
Block block2 = new Block(2, blockchain.getLatestBlock().getHash(), "Data 2");
blockchain.addBlock(block2);
Block block3 = new Block(3, blockchain.getLatestBlock().getHash(), "Data 3");
blockchain.addBlock(block3);
System.out.println("Blockchain is valid: " + blockchain.isChainValid());
}
}
```

EXPECTED OUTPUT:

```
Block mined: 0000074bec710e3fea7ae3468ae45ac5362081b6e857e4e00ec0b9740df5d3e1
Block mined: 0000fc1ebba78d5a752f796d47d65718493af3eec8b84a08021661980af755a1
Block mined: 0000b2d8a402174ef0340addd6935814505e4c8a76cd45514d3de6a40db9e71e
Block mined: 0000e3405e36eff138657139b3ae695a5f399b857564fb5bfeaaed850c4f20b2
Blockchain is valid: true
```

OUTPUT:

WEEK:4

AIM : CreatingERC20 token

An ERC20 token is a standard used for creating and issuing smart contracts on the Ethereum blockchain. Smart contracts can then be used to create smart property or tokenized assets that people can invest in. ERC stands for "Ethereum request for comment," and the ERC20 standard was implemented in 2015

SOURCE CODE:

```
import java.util.HashMap;
import java.util.Map;
public class ERC20Token {
private String name;
private String symbol;
private int decimals;
private Map<String, Integer> balances;
public ERC20Token(String name, String symbol, int decimals) {
this.name = name;
this.symbol = symbol;
this.decimals = decimals;
this.balances = new HashMap<>();
}
public void transfer(String from, String to, int amount) {
int balance = balances.getOrDefault(from, 0);
if (balance < amount) {
System.out.println("Insufficient balance");
return;
}
balances.put(from, balance - amount);
balances.put(to, balances.getOrDefault(to, 0) + amount);
System.out.println("Transfer successful");
}
public int balanceOf(String address) {
return balances.getOrDefault(address, 0);
}
```

```
public String getName() {
return name;  }
public String getSymbol() {
return symbol;  }
public int getDecimals() {
return decimals;  }
public static void main(String[] args) {
ERC20Token token = new ERC20Token("MyToken", "MTK", 18);
// Set initial balances
token.balances.put("Alice", 1000);
token.balances.put("Bob", 500);
token.balances.put("Charlie", 200);
// Perform some transfers
token.transfer("Alice", "Bob", 200);
token.transfer("Charlie", "Alice", 100);
token.transfer("Bob", "Charlie", 50);
// Print final balances
System.out.println("Alice balance: " + token.balanceOf("Alice"));
System.out.println("Bob balance: " + token.balanceOf("Bob"));
System.out.println("Charlie balance: " + token.balanceOf("Charlie"));
}}
```

EXPECTED OUTPUT:

```
Transfer successful
Transfer successful
Transfer successful
Alice balance: 900
Bob balance: 650
Charlie balance: 150
```

OUTPUT:

WEEK:5

AIM: Java code to implement blockchain in Merkle Trees

Merkle trees is an implementation of binary trees where each non-leaf node is a hash of the two child nodes. The leaves can either be the data itself or a hash/signature of the data.

Usages:

Merkle tree(Hash tree) is used to verify any kind of data stored, handled and transferred in and between computers.

Currently, the main use of Merkle tree is to make sure that data blocks received from other peers in a peer-to-peer network are received undamaged and unaltered, and even to check that the other peers do not lie and send fake blocks.

Merkle tree is used in git, Amazon's Dynamo, Cassandra as well as BitCoin.

SOURCE CODE:

```
import java.util.ArrayList;
import java.util.List;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
class MerkleTree {
private List<String> transactions;
private String root;
public MerkleTree(List<String> transactions) {
this.transactions = transactions;
this.root = buildTree();
}
private String buildTree() {
List<String> level = new ArrayList<>(transactions);
while (level.size() > 1) {
List<String>nextLevel = new ArrayList<>();
for (int i = 0; i <level.size(); i += 2) {
String left = level.get(i);
String right = (i + 1 <level.size()) ? level.get(i + 1) : "";
String combined = left + right;
String hash = calculateHash(combined);
nextLevel.add(hash);
}
```

```
level = nextLevel;
}
return level.get(0);
}
private String calculateHash(String input) {
try {
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hashBytes = digest.digest(input.getBytes());
StringBuilder hexString = new StringBuilder();
for (byte hashByte : hashBytes) {
String hex = Integer.toHexString(0xff &hashByte);
if (hex.length() == 1)
hexString.append('0');
hexString.append(hex);
}
return hexString.toString();
}
catch (NoSuchAlgorithmException e) {
e.printStackTrace();
return null;
}}
public String getRoot() {
return root;
}}
public class Blockchain1 {
private List<MerkleTree> blocks;
public Blockchain1() {
this.blocks = new ArrayList<>();
}
public void addBlock(List<String> transactions) {
MerkleTree merkleTree = new MerkleTree(transactions);
blocks.add(merkleTree);
}
public String getBlockRoot(int blockIndex) {
if (blockIndex>= 0 &&blockIndex<blocks.size()) {
```

```
MerkleTree merkleTree = blocks.get(blockIndex);
return merkleTree.getRoot();
}
return null;
}
public static void main(String[] args) {
Blockchain1 blockchain = new Blockchain1();
List<String> transactions1 = new ArrayList<>();
transactions1.add("Transaction 1");
transactions1.add("Transaction 2");
transactions1.add("Transaction 3");
blockchain.addBlock(transactions1);
List<String> transactions2 = new ArrayList<>();
transactions2.add("Transaction 4");
transactions2.add("Transaction 5");
blockchain.addBlock(transactions2);
String root1 = blockchain.getBlockRoot(0);
System.out.println("Block 1 Root: " + root1);
String root2 = blockchain.getBlockRoot(1);
System.out.println("Block 2 Root: " + root2);
}
}
```

EXPECTED OUTPUT:

```
Block 1 Root: b120c9e964f9e99146b13316ae3b800b4995e1f4b57882f24d452a758e690ba3
Block 2 Root: 4374f160912ba3d9a66adf4aee6b9c1a77d9baeb12d1fecce5a36d59f19011cf
```

OUTPUT:

WEEK:6

AIM: Java Code to implement Mining using block chain

Blockchain is a budding technology that has tremendous scope in the coming years. Blockchain is the modern technology that stores data in the form of block data connected through cryptography and cryptocurrencies such as Bitcoin. It was introduced by **Stuart Haber and W. Scott Tornetta in 1991**. It is a linked list where the nodes are the blocks in the Blockchain, and the references are hashes of the previous block in the chain. References are cryptographic hashes when dealing with link lists. The references are just basically objects. So every single node will store another node variable, and it will be the reference to the next node. In this case, the references are cryptographic hashes.

Blockchain uses hash pointers to reference the previous node in a long list. We assign a hash to every single node because this is how we can identify them

SOURCE CODE:

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
class Block  {
private int index;
private long timestamp;
private String previousHash;
private String hash;
private int nonce;
private List<Transaction> transactions;
public Block(int index, long timestamp, String previousHash, List<Transaction> transactions) {
this.index = index;
this.timestamp = timestamp;
this.previousHash = previousHash;
this.transactions = transactions;
this.nonce = 0;
this.hash = calculateHash();
}
public String calculateHash() {
```

```
try {
MessageDigest digest = MessageDigest.getInstance("SHA-256");
String data = index + timestamp + previousHash + nonce + transactions.toString();
byte[] hashBytes = digest.digest(data.getBytes(StandardCharsets.UTF_8));
StringBuilder hexString = new StringBuilder();
for (byte hashByte : hashBytes) {
String hex = Integer.toHexString(0xff &hashByte);
if (hex.length() == 1)
hexString.append('0');
hexString.append(hex);
}
return hexString.toString();
}
catch (NoSuchAlgorithmException e) {
e.printStackTrace();
}
return null;
}
public void mineBlock(int difficulty) {
String target = new String(new char[difficulty]).replace('\0', '0');
while (!hash.substring(0, difficulty).equals(target)) {
nonce++;
hash = calculateHash();
}
System.out.println("Block mined: " + hash);
}
public String getHash() {
return hash;
}
public String getPreviousHash() {
return previousHash;
}}
class Transaction {
private String from;
private String to;
```

```
private double amount;
public Transaction(String from, String to, double amount) {
this.from = from;
this.to = to;
this.amount = amount;
}
@Override
public String toString() {
return from + "->" + to + ": " + amount;
}}
class Blockchain {
private List<Block> chain;
private int difficulty;
public Blockchain(int difficulty) {
this.chain = new ArrayList<>();
this.difficulty = difficulty;
createGenesisBlock();
}
private void createGenesisBlock() {
List<Transaction> transactions = new ArrayList<>();
transactions.add(new Transaction("Genesis", "Alice", 100));
Block genesisBlock = new Block(0, System.currentTimeMillis(), "0", transactions);
genesisBlock.mineBlock(difficulty);
chain.add(genesisBlock);
}
public void addBlock(Block block) {
block.mineBlock(difficulty);
chain.add(block);
}
public boolean isChainValid() {
for (int i = 1; i <chain.size(); i++) {
Block currentBlock = chain.get(i);
Block previousBlock = chain.get(i - 1);
if (!currentBlock.getHash().equals(currentBlock.calculateHash()))
return false;
```

```
if (!currentBlock.getPreviousHash().equals(previousBlock.getHash()))
return false;
}
return true;
}
public Block getLastBlock() {
return chain.get(chain.size() - 1);
}}
public class BlockchainMiningExample {
public static void main(String[] args) {
// Create a blockchain with difficulty 4
Blockchain blockchain = new Blockchain(4);
// Create some transactions and add them to a block
List<Transaction> transactions = new ArrayList<>();
transactions.add(new Transaction("Alice", "Bob", 10.0));
transactions.add(new Transaction("Charlie", "Alice", 5.0));
Block block1 = new Block(1, System.currentTimeMillis(), blockchain.getLastBlock().getHash(),
transactions);
// Add the block to the blockchain
blockchain.addBlock(block1);
// Create another block with different transactions
List<Transaction> transactions2 = new ArrayList<>();
transactions2.add(new Transaction("Bob", "Charlie", 3.0));
transactions2.add(new Transaction("Alice", "Bob", 2.0));
Block block2 = new Block(2, System.currentTimeMillis(), blockchain.getLastBlock().getHash(),
transactions2);
// Add the second block to the blockchain
blockchain.addBlock(block2);
// Validate the blockchain
System.out.println("Is blockchain valid? " + blockchain.isChainValid());
}}
```

EXPECTED OUTPUT:

```
Block mined: 0000837341c2a7637174b2f6a409bae96fcd0704a6c7f261e09ffe3651240989
Block mined: 00003d574812b87c154063964 7c5d4d97c68fcc6d5ff54c35a8a119f2f9e646c
Block mined: 0000e094e35e0a71d34fb0032cc520989788109aa8c998d23f43a1274335132a
Is blockchain valid? true
```

OUTPUT:

WEEK:7

AIM: Java Code to implement peer-to-peer using block chain

Earlier, we made a single blockchain. Now we're going to make a set of them and get them talking to one another. The real point of the blockchain is a distributed system of verification. We can add blocks from any nodes and eventually it gets to peer nodes so everyone agrees on what the blockchain looks like. There is one problem that comes up right away: Each node is two services, plus a MongoDB and a Kafka message bus that all need to talk to one another. We'll be working on a node service that will allow the nodes to work with one another. This will get input from two places, a restful interface that allows you to add and list the nodes connected, and a message bus provided by Kafka that notifies the node service of changes in the local blockchain that need to be broadcast to the peer nodes.

SOURCE CODE:

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
class Block  {
private int index;
private long timestamp;
private String previousHash;
private String hash;
private int nonce;
private List<Transaction> transactions;
public Block(int index, long timestamp, String previousHash, List<Transaction> transactions) {
this.index = index;
this.timestamp = timestamp;
this.previousHash = previousHash;
this.transactions = transactions;
this.nonce = 0;
this.hash = calculateHash();
}
public String calculateHash() {
try {
```

```
MessageDigest digest = MessageDigest.getInstance("SHA-256");
String data = index + timestamp + previousHash + nonce + transactions.toString();
byte[] hashBytes = digest.digest(data.getBytes(StandardCharsets.UTF_8));
StringBuilder hexString = new StringBuilder();
for (byte hashByte : hashBytes) {
String hex = Integer.toHexString(0xff &hashByte);
if (hex.length() == 1)
hexString.append('0');
hexString.append(hex);
}
return hexString.toString();
} catch (NoSuchAlgorithmException e) {
e.printStackTrace();
}
return null;
}
public void mineBlock(int difficulty) {
String target = new String(new char[difficulty]).replace('\0', '0');
while (!hash.substring(0, difficulty).equals(target)) {
nonce++;
hash = calculateHash();
}
System.out.println("Block mined: " + hash);
}
public int getIndex() {
return index;
}
public long getTimestamp() {
return timestamp;
}
public String getPreviousHash() {
return previousHash;
}
public String getHash() {
return hash;
```

```
}
public int getNonce() {
return nonce;
}
public List<Transaction>getTransactions() {
return transactions;
}}
class Transaction {
private String from;
private String to;
private double amount;
public Transaction(String from, String to, double amount) {
this.from = from;
this.to = to;
this.amount = amount;
}
public String getFrom() {
return from;
}
public String getTo() {
return to;
}
public double getAmount() {
return amount;
}
@Override
public String toString() {
return from + "->" + to + ": " + amount;
}}
class Blockchain {
private List<Block> chain;
private int difficulty;
public Blockchain(int difficulty) {
this.chain = new ArrayList<>();
this.difficulty = difficulty;
```

```
createGenesisBlock();
}
private void createGenesisBlock() {
List<Transaction> transactions = new ArrayList<>();
transactions.add(new Transaction("Genesis", "Alice", 100));
Block genesisBlock = new Block(0, System.currentTimeMillis(), "0", transactions);
genesisBlock.mineBlock(difficulty);
chain.add(genesisBlock);
}
public void addBlock(Block block) {
block.mineBlock(difficulty);
chain.add(block);
}
public boolean isChainValid() {
for (int i = 1; i <chain.size(); i++) {
Block currentBlock = chain.get(i);
Block previousBlock = chain.get(i - 1);
if (!currentBlock.getHash().equals(currentBlock.calculateHash()))
return false;
if (!currentBlock.getPreviousHash().equals(previousBlock.getHash()))
return false;
}
return true;
}
public List<Block>getChain() {
return chain;
}
public Block getLastBlock() {
return chain.get(chain.size() - 1);
}}
class Node {
private Blockchain blockchain;
private List<Transaction>pendingTransactions;
public Node(Blockchain blockchain) {
this.blockchain = blockchain;
```

```
this.pendingTransactions = new ArrayList<>();
}
public void minePendingTransactions() {
Block newBlock = new Block(
blockchain.getLastBlock().getIndex() + 1,
System.currentTimeMillis(),
blockchain.getLastBlock().getHash(),pendingTransactions);
blockchain.addBlock(newBlock);
pendingTransactions.clear();
}
public void createTransaction(Transaction transaction) {
pendingTransactions.add(transaction);
}
public Blockchain getBlockchain() {
return blockchain;
}
public List<Transaction>getPendingTransactions() {
return pendingTransactions;
}}
public class PeerToPeerBlockchain {
public static void main(String[] args) {
// Create a blockchain with difficulty 4
Blockchain blockchain = new Blockchain(4);
// Create two nodes
Node node1 = new Node(blockchain);
Node node2 = new Node(blockchain);
// Node 1 creates a transaction
Transaction transaction1 = new Transaction("Alice", "Bob", 10.0);
node1.createTransaction(transaction1);
// Node 2 creates a transaction
Transaction transaction2 = new Transaction("Bob", "Charlie", 5.0);
node2.createTransaction(transaction2);
// Node 1 mines the pending transactions
node1.minePendingTransactions();
// Node 2 mines the pending transactions
```

```
node2.minePendingTransactions();
// Validate the blockchain
System.out.println("Is blockchain valid? " + blockchain.isChainValid());
}}
```

EXPECTED OUTPUT:

```
Block mined: 0000ca2a782ba1ee2eeb3cb3dbc2b9c50a51c9427348d922c5659adc63923419
Block mined: 00006bad081f47fcb42c2952e4a6d03c32e42bccd23c6a7764e0bd1f44805bbc
Block mined: 0000bbaedefd1d7ea417b85dba9b3885d4a57838d3f3182b2833c4cc5f0b90b7
Is blockchain valid? false
```

OUTPUT:

WEEK:8

AIM: creating a Crypto-currency Wallet.

There are four basic steps.

1. Choose the type of wallet.
2. Sign up for an account, buy the device or download the software needed.
3. Set up security features, including a recovery phrase.
4. Purchase cryptocurrency or transfer coins from another wallet or exchange.

Types of crypto wallets:

There are three basic types of wallets for virtual currency.

One option is a software wallet or hot wallet that stores your crypto on an internet-connected device that you own.

Another option to consider with added security is a cold wallet, a specialized piece of hardware that keeps your crypto offline.

Custodial wallets, which leave your crypto in the control of a company you trust, such as a crypto exchange, are another storage method to consider.

SOURCE CODE:

```
import java.security.*;
import java.security.spec.ECGenParameterSpec;
public class CryptoWallet {
private PrivateKey privateKey;
private PublicKey publicKey;
public CryptoWallet() {
generateKeyPair();
}
public void generateKeyPair() {
try {
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC");
SecureRandom random = SecureRandom.getInstanceStrong();
ECGenParameterSpec ecSpec = new ECGenParameterSpec("spec256k1");
keyGen.initialize(ecSpec, random);
KeyPair keyPair = keyGen.generateKeyPair();
privateKey = keyPair.getPrivate();
publicKey = keyPair.getPublic();
} catch (Exception e) {
```

```
e.printStackTrace();
}}
public static void main(String[] args) {
CryptoWallet wallet = new CryptoWallet();
System.out.println("Private Key: " + wallet.privateKey);
System.out.println("Public Key: " + wallet.publicKey);
}}
```

EXPECTED OUTPUT:

```
java.security.InvalidAlgorithmParameterException: Unknown curve name: spec256k1
        at jdk.crypto.ec/sun.security.ec.ECKeyPairGenerator.initialize(ECKeyPair
Generator.java:103)
        at java.base/java.security.KeyPairGenerator$Delegate.initialize(KeyPairG
enerator.java:699)
        at CryptoWallet.generateKeyPair(CryptoWallet.java:17)
        at CryptoWallet.<init>(CryptoWallet.java:9)
        at CryptoWallet.main(CryptoWallet.java:28)
Private Key: null
Public Key: null
```

OUTPUT: