

# Block chain Lab Manual

## TABLE OF CONTENTS

S.No	Name of the Program	Page no
1	Creating Merkle tree	1
2	Creation of Block	6
3	Block chain Implementation Programming code.	11
4	Creating ERC20 token	16
5	Java code to implement blockchain in Merkle Trees.	20
6	Java Code to implement Mining using block chain.	25
7	Java Code to implement peer-to-peer using block chain.	32
8	Creating Crypto-currency Wallet	40
9	Micro Project.	

**Practical NO. 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) {
```

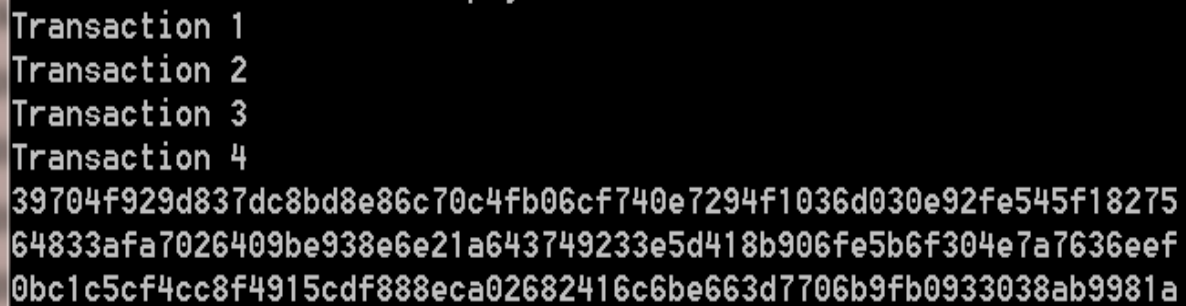
```
hexString.append('0');
}
hexString.append(hex);
}
return hexString.toString();
}
catch (NoSuchAlgorithmException e)
{ e.printStackTrace();
}
return null;
}

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);  
}}
```

**OUTPUT:**A screenshot of a terminal window with a black background and white text. It displays the output of a Java program. The first four lines are "Transaction 1", "Transaction 2", "Transaction 3", and "Transaction 4". The fifth line is a long hexadecimal string representing a Merkle tree hash.

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

**Practical No. 2****AIM : Creation of Block**

Blocks are data structures within the blockchain database, where transaction data in a cryptocurrency 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 validators successfully validate the encrypted information in the blockheader, 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();
}}
```

**OUTPUT:**

```
Block mined: 0afa2aa66eacfd6cc776c8cd7856e354d52303a699bed38560de49efebd9cce3
```

**Practical No.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());
}
```

**OUTPUT:**

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

**Practical No. 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);
```

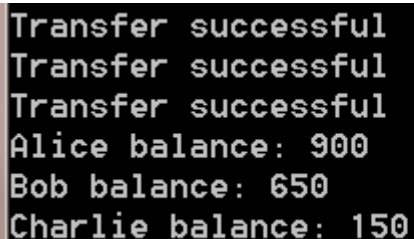
```
public String getName()
{ return name; }

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"));
}}
```

**OUTPUT:**



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

**Practical no.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);
}
}
```

**OUTPUT:**

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

**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 Tormetta 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());  
}}
```

**OUTPUT:**

```
Block mined: 0000837341c2a7637174b2f6a409bae96fcd0704a6c7f261e09ffe3651240989  
Block mined: 00003d574812b87c1540639647c5d4d97c68fcc6d5ff54c35a8a119f2f9e646c  
Block mined: 0000e094e35e0a71d34fb0032cc520989788109aa8c998d23f43a1274335132a  
Is blockchain valid? true
```

**Practical no. 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().getInde
x() + 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());
}}
```

**OUTPUT:**

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

**Practical no.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);
```

```
}
```

```
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
```

























