Program to Create block :

```
import java.security.MessageDigest;
import java.util.Date;

// Block Class
class Block {
public int index;
public long timestamp;
public String data;
public String previousHash;
public String hash;
public int nonce;

// Constructor for Block
public Block(int index, String data, String previousHash) {
this.index = index;
q this.timestamp = new Date().getTime();
this.data = data;
this.previousHash = previousHash;
this.nonce = 0;
this.hash = calculateHash(); // calculate hash during block creation
}

// Method to calculate hash of the block
public String calculateHash() {
String dataToHash = index + Long.toString(timestamp) + data + previousHash +
Integer.toString(nonce);
return applySHA256(dataToHash);
}

// Method for SHA-256 hashing
private static String applySHA256(String input) {
try {
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hashBytes = digest.digest(input.getBytes());
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);
}
}

// Method to mine a block (for proof of work)
public void mineBlock(int difficulty) {
String target = new String(new char[difficulty]).replace('\0', '0'); // e.g., "0000"
while (!hash.substring(0, difficulty).equals(target)) {
nonce++;
hash = calculateHash();
}
System.out.println("Block mined: " + hash);
}
}

// Blockchain Class to store and manage blocks
public class FixedBlockchain {
private static FixedBlockchain instance;
private static final int DIFFICULTY = 4; // Number of leading zeros required for the hash (Proof of
Work)
private static final int MAX_BLOCKS = 5; // Set the maximum number of blocks to mine

// Genesis Block
private Block genesisBlock;
private Block lastBlock;

// Constructor for Blockchain
private FixedBlockchain() {
// Create the genesis block (first block)
genesisBlock = new Block(0, "Genesis Block", "0");
genesisBlock.mineBlock(DIFFICULTY);
lastBlock = genesisBlock;
}

// Singleton pattern to ensure only one instance of the blockchain
public static FixedBlockchain getInstance() {
if (instance == null) {
instance = new FixedBlockchain();
}
return instance;
}

// Add a new block to the blockchain
public void addBlock(String data) {
Block newBlock = new Block(lastBlock.index + 1, data, lastBlock.hash);
newBlock.mineBlock(DIFFICULTY); // Mine the block
lastBlock = newBlock;
```

```
System.out.println("Block added to blockchain: " + newBlock.hash);
}

// Method to mine a fixed number of blocks
public void startMining() {
for (int i = 0; i < MAX_BLOCKS; i++) {
String data = "Block #" + i + " - Data";
addBlock(data);
}
}

public static void main(String[] args) {
FixedBlockchain blockchain = FixedBlockchain.getInstance();
blockchain.startMining(); // Start mining a fixed number of blocks (e.g., 5 blocks)
}
}
```

OUTPUT:

```
computer@computer-thinkcentre-neo-50t-gen-3:~/Desktop/Jiten$ java FixedBlockchain
Block mined: 0000abcca9604f2392bd54b96e1a1224a03803837dcbccb99f9297f054d05605
Block mined: 0000cbe9570df6de91d4f6d05a70d811e882a87b213541f4025ae7c5a195c8aa
Block added to blockchain:
0000cbe9570df6de91d4f6d05a70d811e882a87b213541f4025ae7c5a195c8aa
Block mined: 00006f3ca3618dd1591dfeacc97844ae8972dc4fdc924e75425ab6029fe8b9bc
Block added to blockchain:
00006f3ca3618dd1591dfeacc97844ae8972dc4fdc924e75425ab6029fe8b9bc
Block mined: 0000df78c9b183dde3e83dcd9c65301829e1728c6b1e55636f8e1719e726896e
Block added to blockchain:
0000df78c9b183dde3e83dcd9c65301829e1728c6b1e55636f8e1719e726896e
Block mined: 00004f6f4b96b8bca1dc70d850dd15f549550692590779cc955f27b72812580b
Block added to blockchain:
00004f6f4b96b8bca1dc70d850dd15f549550692590779cc955f27b72812580b
Block mined: 000077528865f11cc6f52759a55b560baba224f98b9f7194e4d4e41de5f9ec8d
Block added to blockchain:
000077528865f11cc6f52759a55b560baba224f98b9f7194e4d4e41de5f9ec8d
computer@computer-thinkcentre-neo-50t-gen-3:~/Desktop/Jiten$ ^C
```

PROGRAMME FOR MERKLE TREE :

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
```

```
public class MerkleTree {

// Node class to represent each element in the tree
static class MerkleNode {
String hash;
MerkleNode left, right;

MerkleNode(String hash) {
this.hash = hash;
this.left = null;
this.right = null;
}

// Method to print the tree structure
public void printTree(String prefix) {
System.out.println(prefix + hash);
if (left != null) {
left.printTree(prefix + "L--");
}
if (right != null) {
right.printTree(prefix + "R--");
}
}
}

// Helper function to calculate the SHA-256 hash of a string
private static String getHash(String input) {
try {
MessageDigest digest = MessageDigest.getInstance("SHA-256");
byte[] hashBytes = digest.digest(input.getBytes());
StringBuilder hexString = new StringBuilder();
for (byte b : hashBytes) {
hexString.append(String.format("%02x", b));
}
return hexString.toString();
} catch (NoSuchAlgorithmException e) {
e.printStackTrace();
return null;
}
}

// Function to build the Merkle Tree
public static MerkleNode buildMerkleTree(List<String> data) {
// Step 1: Create leaf nodes (hash of data)
List<MerkleNode> nodes = new ArrayList<>();
for (String datum : data) {
```

```
nodes.add(new MerkleNode(getHash(datum)));
}

// Step 2: Build the tree by pairing nodes
while (nodes.size() > 1) {
List<MerkleNode> newLevel = new ArrayList<>();
for (int i = 0; i < nodes.size(); i += 2) {
if (i + 1 < nodes.size()) {
// Combine the hashes of the two child nodes
String combinedHash = getHash(nodes.get(i).hash + nodes.get(i + 1).hash);
MerkleNode parent = new MerkleNode(combinedHash);
parent.left = nodes.get(i);
parent.right = nodes.get(i + 1);
newLevel.add(parent);
} else {
// If there's an odd number of nodes, promote the last one to the next level
newLevel.add(nodes.get(i));
}
}
nodes = newLevel;
}

// Return the root node of the tree
return nodes.get(0);
}

// Function to get the Merkle Root of the tree
public static String getMerkleRoot(List<String> data) {
MerkleNode root = buildMerkleTree(data);
return root.hash;
}

// Example of using the Merkle Tree
public static void main(String[] args) {
List<String> data = new ArrayList<>();
data.add("data1");
data.add("data2");
data.add("data3");
data.add("data4");

// Build the Merkle tree and print it
MerkleNode root = buildMerkleTree(data);
System.out.println("Merkle Tree Structure:");
root.printTree("");

// Print the Merkle Root
String merkleRoot = root.hash;
```

```
System.out.println("\nMerkle Root: " + merkleRoot);
}
}
```

OUTPUT:

```
computer@computer-thinkcentre-neo-50t-gen-3:~/Desktop/Jiten$ java MerkleTree
Merkle Tree Structure:
51a0d54f81dcc317ea21d2125c65d796eac64e7c52b886d40388cf1f1abf93eb
L--7a598b35dcbb2b6c7b45ffc1e4152a1f822ef41f68fff3a1b457d057629d89ec
L--L--5b41362bc82b7f3d56edc5a306db22105707d01ff4819e26faef9724a2d406c9
L--R--d98cf53e0c8b77c14a96358d5b69584225b4bb9026423cbc2f7b0161894c402c
R--23431736aac0ab2cab427b40cae8253bf66e3fb5721f34696cf54730aefce451
R--L--f60f2d65da046fcaaf8a10bd96b5630104b629e111aff46ce89792e1caa11b18
R--R--02c6edc2ad3e1f2f9a9c8fea18c0702c4d2d753440315037bc7f84ea4bba2542

Merkle Root: 51a0d54f81dcc317ea21d2125c65d796eac64e7c52b886d40388cf1f1abf93eb
```

CREATED BY:
1. Piyush Badwaik
2.Vansh Hinge
3. Yashpal Chandewar
4. Khilesh Dhekwar