BT Internal - I

Week 1: Creating Merkle tree

```
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
```

Week 2: Creation of Block

```
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

Week 3: Blockchain Implementation Programming 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
```

Week 4: CreatingERC20 token

```
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
```