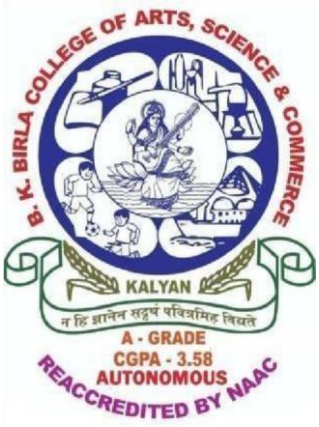
B. K. BIRLA COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS), KALYAN

DEPARTMENT OF INFORMATION TECHNOLOGY



Student Name:	**Aditya Arun Deo**
Student ID:	**5187109**
Class:	**MSc IT (CC)**
Subject:	**Blockchain Technology**

B. K. BIRLA COLLEGE OF ARTS, SCIENCE & COMMERCE (AUTONOMOUS), KALYAN

(Affiliated to University of Mumbai)

KALYAN-MAHARASHTRA-421301

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that Mr **Aditya Arun Deo** bearing Seat. No: **(23258706)**, in class **MSc IT CC** has successfully completed practical of the subject **Blockchain Technology.**

Teacher's Signature: _____ **Date: 05/06/2025**

Place: Kalyan **College Seal**

INDEX

SR. NO	PRACTICAL NAME	SIGNATURE
1	Creating Merkle Tree	
2	Creation of Block	
3	Blockchain implementation	
4	Creating ERC20 token	
5	Blockchain implementation using Merkle Trees	
6	Mining in Blockchain	
7	Peer-to-Peer implementation using Blockchain	
8	Creating Crypto-currency Wallet	

Practical: 1

AIM: Creating Merkle tree Merkle Tree

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();

        System.out.println("Merkle Tree:");
        for (String hash : tree) {
            System.out.println(hash);
        }

        System.out.println("\nMerkle Root:");
        System.out.println(tree.get(tree.size() - 1));
    }
}
```

Output:

```
Output
Merkle Tree:
Transaction 1
Transaction 2
Transaction 3
Transaction 4
39704f929d837dc8bd8e86c70c4fb06cf740e7294f1036d030e92fe545f18275
64833afa7026409be938e6e21a643749233e5d418b906fe5b6f304e7a7636eef
0bc1c5cf4cc8f4915cdf888eca02682416c6be663d7706b9fb0933038ab9981a

Merkle Root:
0bc1c5cf4cc8f4915cdf888eca02682416c6be663d7706b9fb0933038ab9981a

=== Code Execution Successful ===
```

Practical: 2

AIM: Creation of Block

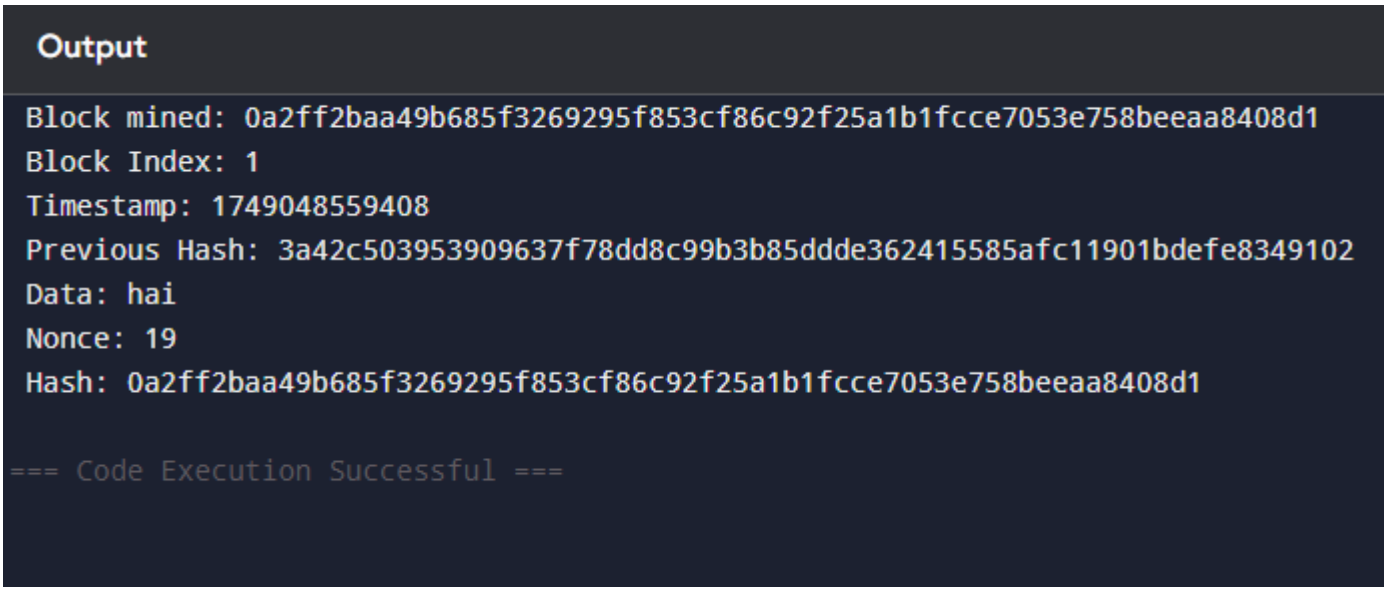
Source Code:

```
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 void displayBlock() {
    System.out.println(" Block Index: " + index);
    System.out.println(" Timestamp: " + timestamp);
    System.out.println(" Previous Hash: " + previousHash);
    System.out.println(" Data: " + data);
    System.out.println(" Nonce: " + nonce);
```

```
        System.out.println(" Hash: " + hash);
    }

    public static void main(String[] args) {
        Block b = new Block(1,
"3a42c503953909637f78dd8c99b3b85ddde362415585afc11901bdefe8349102", "hai");
        b.mineBlock(1);  // Difficulty = 1 for faster execution
        b.displayBlock();
    }
}
```

Output:



Practical: 3

AIM: Block chain Implementation Programming code

Source Code:

```
import java.util.*;
import java.security.*;

public class SimpleBlockchain {
   static class Block {
      String data, previousHash, hash;
      int nonce;

      Block(String data, String previousHash) {
         this.data = data;
         this.previousHash = previousHash;
         this.hash = calculateHash();
      }

      String calculateHash() {
         try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            String input = data + previousHash + nonce;
            byte[] hashBytes = digest.digest(input.getBytes());
            StringBuilder hex = new StringBuilder();
            for (byte b : hashBytes) hex.append(String.format("%02x", b));
            return hex.toString();
         } catch (Exception e) {
            return null;
         }
      }
```

```
    void mineBlock(int diff) {
        String target = "0".repeat(diff);
        while (!hash.startsWith(target)) {
            nonce++;
            hash = calculateHash();
        }
        System.out.println("Mined: " + hash);
    }
}

public static void main(String[] args) {
    int difficulty = 2;
    List<Block> blockchain = new ArrayList<>();

    Block genesis = new Block("Genesis Block", "0");
    genesis.mineBlock(difficulty);
    blockchain.add(genesis);

    Block second = new Block("Data Block 1", genesis.hash);
    second.mineBlock(difficulty);
    blockchain.add(second);

    System.out.println("Valid: " + blockchain.get(1).previousHash.equals(blockchain.get(0).hash));
}
}
```

Output:

```
Output
Mined: 002f469fc7bfe747e889362575910b42609e22b5baea5890034aa9b6701ded62
Mined: 000bc336fc78ef56e1431e2b16cd5214121f0d72a3ea471a747ee92fc7601f93
Valid: true

=== Code Execution Successful ===
```

Practical: 4

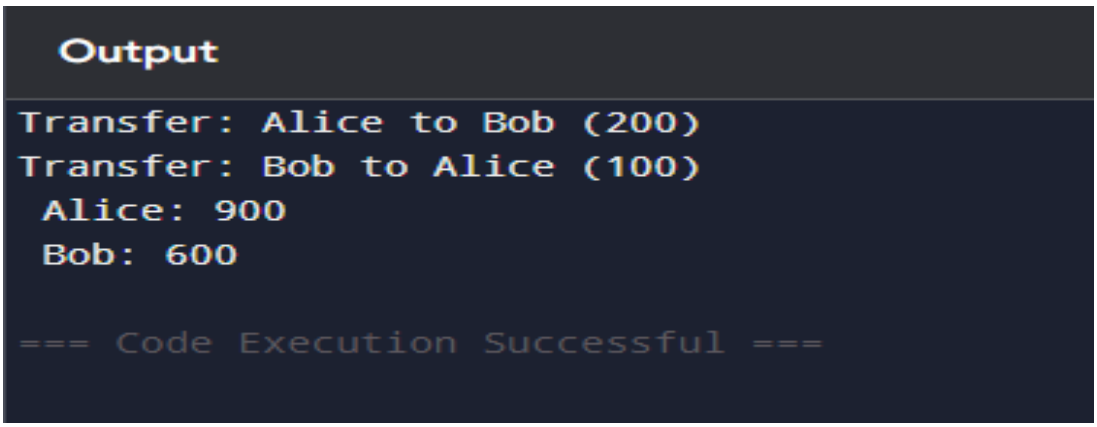
AIM: CreatingERC20 token

Source Code:

```
import java.util.*;

public class ERC20Token {
    String name, symbol;
    int decimals;
    Map<String, Integer> balances = new HashMap<>();

    ERC20Token(String name, String symbol, int decimals) {
        this.name = name;
        this.symbol = symbol;
        this.decimals = decimals;
    }

    void transfer(String from, String to, int amount) {
        int bal = balances.getOrDefault(from, 0);
        if (bal < amount) {
            System.out.println("Insufficient balance");
            return;
        }
        balances.put(from, bal - amount);
        balances.put(to, balances.getOrDefault(to, 0) + amount);
        System.out.println("Transfer: " + from + " to " + to + " (" + amount + ")");
    }

    int balanceOf(String user) {
        return balances.getOrDefault(user, 0);
```

```
    }

    public static void main(String[] args) {
        ERC20Token token = new ERC20Token("MyToken", "MTK", 18);

        token.balances.put("Alice", 1000);
        token.balances.put("Bob", 500);

        token.transfer("Alice", "Bob", 200);
        token.transfer("Bob", "Alice", 100);

        System.out.println(" Alice: " + token.balanceOf("Alice"));
        System.out.println(" Bob: " + token.balanceOf("Bob"));
    }
}
```

Output:



Practical: 5

AIM: Java code to implement blockchain in Merkle Trees

Source Code:

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

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) : "";
                nextLevel.add(calculateHash(left + right));
            }
            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 b : hashBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public String getRoot() {
        return root;
    }
}

public class Main {
    private List<MerkleTree> blocks;

    public Main() {
        blocks = new ArrayList<>();
    }

    public void addBlock(List<String> transactions) {
```

```
        MerkleTree merkle = new MerkleTree(transactions);
        blocks.add(merkle);
    }

    public String getBlockRoot(int index) {
        if (index >= 0 && index < blocks.size()) {
            return blocks.get(index).getRoot();
        }
        return null;
    }

    public static void main(String[] args) {
        Main blockchain = new Main();

        List<String> tx1 = Arrays.asList("Tx1", "Tx2", "Tx3");
        List<String> tx2 = Arrays.asList("Tx4", "Tx5");

        blockchain.addBlock(tx1);
        blockchain.addBlock(tx2);

        System.out.println("Block 1 Merkle Root: " + blockchain.getBlockRoot(0));
        System.out.println("Block 2 Merkle Root: " + blockchain.getBlockRoot(1));
    }
}
```

Output:

```
Output
Block 1 Merkle Root: a1cda5145539e482c4e898a4fcf183a0235e507af754c144a3480436ade81a0e
Block 2 Merkle Root: 0e72d0c79191e6816a9d0fa90b437fb15797b4d9b357e1c8d8fae05f6c134848

=== Code Execution Successful ===
```

Practical: 6

AIM: Java Code to implement Mining using block chain

Source Code:

```
import java.util.*;
import java.security.*;

class Block {
    int index, nonce;
    long timestamp;
    String prevHash, hash, data;

    public Block(int index, String prevHash, String data) {
        this.index = index;
        this.prevHash = prevHash;
        this.data = data;
        this.timestamp = System.currentTimeMillis();
        this.nonce = 0;
        this.hash = calculateHash();
    }

    public String calculateHash() {
        try {
            String input = index + prevHash + data + timestamp + nonce;
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hashBytes = digest.digest(input.getBytes());
            StringBuilder hex = new StringBuilder();
            for (byte b : hashBytes) {
                String h = Integer.toHexString(0xff & b);
                if (h.length() == 1) hex.append('0');
```

```
            hex.append(h);
        }
        return hex.toString();
    } catch (Exception e) { return null; }
  }

  public void mine(int diff) {
    String target = "0".repeat(diff);
    while (!hash.substring(0, diff).equals(target)) {
      nonce++;
      hash = calculateHash();
    }
    System.out.println("Block mined: " + hash);
  }
}

public class Main {
  public static void main(String[] args) {
    int difficulty = 3;
    List<Block> chain = new ArrayList<>();

    Block genesis = new Block(0, "0", "Genesis Block");
    genesis.mine(difficulty);
    chain.add(genesis);

    Block b1 = new Block(1, chain.get(0).hash, "Alice pays Bob 50");
    b1.mine(difficulty);
    chain.add(b1);

    Block b2 = new Block(2, chain.get(1).hash, "Bob pays Charlie 30");
```

```
        b2.mine(difficulty);
        chain.add(b2);

        boolean valid = true;
        for (int i = 1; i < chain.size(); i++) {
            if (!chain.get(i).hash.equals(chain.get(i).calculateHash()) ||
                !chain.get(i).prevHash.equals(chain.get(i - 1).hash)) {
                valid = false;
                break;
            }
        }

        System.out.println("Is blockchain valid? " + valid);
    }
}
```

Output:

```
Output
Block mined: 000ebc9aa142d479dd8a1c5c89d9abefd2340bff0e4ed0503169312940f7cc4e
Block mined: 000d127621130f3304446a438bb6064ca15b48b5113bb0314f3770dbd7b49e3e
Block mined: 000ecd192af5e9215839722fa6d0560d79aeba2f3261d7d135975c18820d52ae
Is blockchain valid? true

=== Code Execution Successful ===
```

Practical: 7

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

Source Code:

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

public class SimpleBlockchain {
   static class Block {
      String prevHash, hash;
      List<String> transactions;
      int nonce = 0;

      Block(String prevHash, List<String> transactions) {
         this.prevHash = prevHash;
         this.transactions = transactions;
         this.hash = calculateHash();
      }

      String calculateHash() {
         try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            String data = prevHash + transactions.toString() + nonce;
            byte[] hashBytes = digest.digest(data.getBytes());
            StringBuilder sb = new StringBuilder();
            for (byte b : hashBytes) sb.append(String.format("%02x", b));
            return sb.toString();
         } catch (Exception e) {
            throw new RuntimeException(e);
         }
```

```
    }

    void mine(int difficulty) {
        String target = "0".repeat(difficulty);
        while (!hash.substring(0, difficulty).equals(target)) {
            nonce++;
            hash = calculateHash();
        }
        System.out.println("Mined: " + hash);
    }
}

List<Block> chain = new ArrayList<>();
int difficulty = 3;

public SimpleBlockchain() {
    chain.add(new Block("0", List.of("Genesis Block")));
    chain.get(0).mine(difficulty);
}

public void addBlock(List<String> transactions) {
    Block prev = chain.get(chain.size() - 1);
    Block newBlock = new Block(prev.hash, transactions);
    newBlock.mine(difficulty);
    chain.add(newBlock);
}

public boolean isValid() {
    for (int i = 1; i < chain.size(); i++) {
        Block curr = chain.get(i);
```

```
            Block prev = chain.get(i - 1);
            if (!curr.hash.equals(curr.calculateHash())) return false;
            if (!curr.prevHash.equals(prev.hash)) return false;
        }
        return true;
    }

    public static void main(String[] args) {
        SimpleBlockchain blockchain = new SimpleBlockchain();

        blockchain.addBlock(List.of("Alice pays Bob 10", "Charlie pays Dave 5"));
        blockchain.addBlock(List.of("Bob pays Eve 3"));

        System.out.println("Blockchain valid? " + blockchain.isValid());
    }
}
```

Output:

```
Output
Mined: 00075275985491929ff5989be00a958ef77b6b1f62be98686c52ad8807d30f83
Mined: 00086a70373fd5f85b594ed4daed056847d36c7d2e06d2df827ee49b96e0123d
Mined: 000cff33c07e9e72cd86697c99df7a6108308d5747468621db9a22645a1273c6
Blockchain valid? true

=== Code Execution Successful ===
```

Practical: 8

AIM: Creating a Crypto-currency Wallet.

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("secp256k1");
            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);
    }
}
```

Output: