Source Code

Bank.js:

*const* { exit } = require('process');

*const* {Node,BST} = require('./BST.js');

*const* read\_sync = require('readline-sync');

*let* tree = new BST();

console.log("\n Welcome to Banking Application!!! \n Please choose the type of service: \n");

*let* key;

do{console.log('1. Create account \n2. Delete Account \n3. Amount Transfer '+

    '\n4. Check Balance \n5. view History  \n6. Exit \n');

 key = read\_sync.question("Enter your choice:");

 switch (key) {

     case '1': //Create account

         createAcount();

         break;

     case '2': //Delete Account

         deleteAccount();

         break;

     case '3': //Amount Transfer

         createTransaction();

         break;

     case '4': //Check Balance

         checkBalance();

         break;

     case '5': //Exit

         viewHistory();

         break;

     case '6': //Exit

         exit(0);

     default:

         console.log("Please choose the correct option between 1 and 6 !!!");

         break;

     }

}while(key<7);

*function* createAcount(){

    do{

        Account\_no = parseInt(read\_sync.question('Enter the account number : '));

        balance = parseInt(read\_sync.question('Enter the amount to deposit(minimum balance: Rs.2500): '));

    }while(balance<2500);

    tree.root = tree.insert(tree.root,new Node(Account\_no,balance));

}

*function* deleteAccount(){

    Account\_no = parseInt(read\_sync.question('Enter the account number : '));

     tree.root = tree.delete(tree.root,Account\_no);

}

*function* createTransaction(){

    sender = parseInt(read\_sync.question('Enter your account number : '));

    receiver = parseInt(read\_sync.question('Enter the receiver\'s account number : '));

    amount = parseInt(read\_sync.question('Enter the amount to deposit: '));

    tree.root = tree.makeTransaction(tree.root,sender,receiver,amount);

}

*function* checkBalance(){

    Account\_no = parseInt(read\_sync.question('Enter the account number : '));

    User = tree.search(tree.root,Account\_no);

    if(User != false){

        console.log("Your Balance is: Rs."+User.balance);

    }

}

BST.js:

*class* Node{

*constructor*(*account\_no*,*balance*){

        this.account\_no = *account\_no*;

        this.balance = *balance*;

        this.left = null;

        this.right = null;

        this.transactions = null

    }

}

*class* Transaction{

*constructor*(*action*){

        this.action = *action*;

        this.next = null;

    }

    linkedListInsert(*head*,*transaction*){

        if(*head* == null){

*head* = *transaction*;

            return *head*;

        }

        else{

            temp = *head*;

            while( temp.next != null){

                temp = temp.next;

            }

            temp.next = *transaction*;

            return *head*;

        }

    }

    displayStatement(*transaction*){

        while(*transaction* != null){

            console.log(*transaction*.action);

*transaction* = *transaction*.next;

        }

    }

}

*class* BST{

*constructor*(){

    this.root = null;

   }

   /////////////////////////////////////////////////////////////////////////////////////

   insert(*rootNode*,*newNode*){

            if(*rootNode* === null){

                    ///creating root node

                    if(this.root === null){

                        this.root = *newNode*;

                        console.log("Account for "+*newNode*.account\_no+" created Successfully!!!");

                        return this.root;

                    }

                    else{  // creating a new leaf node

*rootNode* = *newNode*;

                        console.log("Account for "+*newNode*.account\_no+" created Successfully!!!");

                        return *rootNode*;

                    };

            }

            // checking if the account already exists

            else if (*newNode*.account\_no==*rootNode*.account\_no){

                    console.log("Account no "+*newNode*.account\_no+" already exists !!!");

                    return null;

            }

            // traversing the tree to insert a new node to the left sub tree

            else if(*newNode*.account\_no<*rootNode*.account\_no)

*rootNode*.left = this.insert(*rootNode*.left,*newNode*);

            // traversing the tree to insert a new node to the right sub tree

            else

                rootNode.right = this.insert(rootNode.right,newNode);

            return rootNode;

        }

   //////////////////////////////////////////////////////////////////////////////////////////

   search(*rootNode*,*account\_no*){

        if(rootNode == null){ // when the account is not found

            console.log("Account Number "+account\_no+" does not exists. Please check the account number!!!");

            return false;

        }

        // when the account is found

        else if(this.root.account\_no == account\_no || rootNode.account\_no == account\_no){

            return ((this.root.account\_no == account\_no)?this.root:rootNode);

        }

        else if(account\_no<rootNode.account\_no){  // traversing the left sub tree for the acoount

        return (this.search(rootNode.left,account\_no));

        }

        else{ // traversing the right sub tree for the acoount

            return (this.search(rootNode.right,account\_no));

        }

    }

   ////////////////////////////////////////////////////////////////////////////////////////

   update(*rootNode*,*account\_no*,*balance*,*transaction*){

        if(rootNode == null){  //when the account is not found

            console.log("Account Number "+account\_no+" not present!!!");

            return false;

        }

        // when the account is found

        else if(this.root.account\_no == account\_no || rootNode.account\_no == account\_no){

             if(this.root.account\_no == account\_no){  //for rootNode

                this.root.balance=balance;

                this.root.transactions = transaction;

                return this.root;

            }

            else{

                rootNode.balance = balance;

                rootNode.transactions = transaction;

                return rootNode;

            }

        }

        // traversing the left sub tree for the acoount

        else if(account\_no<rootNode.account\_no){

            rootNode.left = this.search(rootNode.left,account\_no);

            return rootNode;

        }

        else{  // traversing the right sub tree for the acoount

            rootNode.right = this.search(rootNode.right,account\_no);

            return rootNode;

        }

    }

   ////////////////////////////////////////////////////////////////////////////////////////

   delete(*rootNode*,*account\_no*){

        if(rootNode == null){

            console.log("Please check the account number!!!");

            return null;

        }

        if(account\_no<rootNode.account\_no)

            rootNode.left = this.delete(rootNode.left,account\_no);

        else if(account\_no>rootNode.account\_no)

            rootNode.right = this.delete(rootNode.right,account\_no);

        else if(account\_no == rootNode.account\_no){

            if(rootNode.left==null && rootNode.right==null ){

                console.log("Account number "+account\_no+" deleted successfully!!!");

                return null;

            }

            else if(rootNode.left===null || rootNode.right===null){

                ///for deleting leaf nodes and subtrees with a child(left or right)

*let* temp = ((rootNode.left === null)?rootNode.right : rootNode.left);

                console.log("Account number "+account\_no+" deleted successfully!!!");

                return temp;

            }

            else{

                // for subtree with two children

*let* Successor = this.getSuccessor((rootNode.right.left==null?rootNode:rootNode.right));// finding the substitute node

                //Successor.left = rootNode.left;

                //Successor.right = this.delete(rootNode.right,Successor.data);

*let* leftchild = rootNode.left;

*let* rightchild = rootNode.right;

                rootNode = Successor; // updating the fields of the successor

                rootNode.left = leftchild; // updating the subtrees of the new node

                rootNode.right = rightchild;

                console.log("Account number "+account\_no+" deleted successfully!!!");

                return temp;

            }

        }

        else{

            console.log("Please check the account number!!!");

            return;

        }

        return temp;

        }

    getSuccessor(*rootNode*){

        while(*rootNode*.left.left !== null){ // more than one subtree.

*rootNode* = *rootNode*.left;

        }

*let* temp = *rootNode*.left

*rootNode*.left = null; // updating the successor's pointers

        return temp; // returning the successor

    }

    makeTransaction(*rootNode*,*sender*,*receiver*,*amount*){

*let* senderNode = this.search(*rootNode*,*sender*);

*let* receiverNode = this.search(*rootNode*,*receiver*);

        if((senderNode == false && receiverNode == false)){

            console.log("Please check the account number!!!");

            return *rootNode*;

        }

        else if(senderNode.balance-*amount*<2500){

            console.log("Transaction failed due to insufficient balance!!!");

        }

        else{

*let* date = (new *Date*()).toString();

*let* senderHistory = new Transaction("Amount of Rs. "+*amount*+" debited to the account no. "+*receiver*+" on "+ date);

            senderNode.transactions = senderHistory.linkedListInsert(senderNode.transactions,senderHistory);

*let* receiverHistory = new Transaction("Amount of Rs."+*amount*+" credited from the account no. "+*receiver*+" on "+ date);

            receiverNode.transactions = receiverHistory.linkedListInsert(receiverNode.transactions,receiverHistory);

*rootNode* = this.update(*rootNode*,*sender*,senderNode.balance -= *amount*,senderNode.transactions);//

*rootNode* = this.update(*rootNode*,*receiver*,receiverNode.balance += *amount*,receiverNode.transactions);//

            console.log("Transaction successful!!!");

            return *rootNode*;

        }

    }

    viewTransactions(*rootNode*,*sender*){

*let* sender\_acc = this.search(*rootNode*,*sender*);

        if(sender\_acc != false){

            sender\_acc.transactions.displayStatement(sender\_acc.transactions);

        }

    }

}

*module*.*exports* = {Node,BST};