**Advanced Data Structures and Algorithm Project**

**Problem Statement:**

To implement a system for an Admin working at a telephone service company. The system should be able to help the admin to add new customers and to figure out the customers who are low on balance so that they can inform them.

**Data Structure:**

Binary Search Tree

**Solution:**

A Menu driven program which helps the admin perform certain functions like

1. Adding a customer -- **INSERTION**
2. View the list of customers – **IN ORDER TRAVERSAL**
3. Find the customer with Minimum and Maximum Balance – **MINNODE, MAXNODE**
4. Find customers within a given range of Balance to find out the low/high balance customers – **PRINTRANGE**

**Justification:**

BST is chosen over normal binary tree since In-order traversal is not possible in a normal Binary Tree.

**Code:**

package PROJECT;

import java.util.Scanner;

public class ADMIN {

static NODE root;

static class NODE {

NODE leftchild;

NODE rightchild;

long phoneNo;

int Balance;

String Name;

public NODE (long phoneNo, int Balance, String Name){

this.phoneNo = phoneNo;

this.Balance = Balance;

this.Name = Name;

}

}

public void insertion(long phoneNo, int Balance, String Name)

{

NODE n = new NODE(phoneNo, Balance, Name);

if(root == null)

{

root = n;

}

else

{

NODE currentNODE = root;

NODE parentNODE = null;

while(true)

{

parentNODE = currentNODE;

if(Balance < currentNODE.Balance)

{

currentNODE = currentNODE.leftchild;

if(currentNODE == null)

{

parentNODE.leftchild = n;

return;

}

}

else

{

currentNODE = currentNODE.rightchild;

if(currentNODE == null)

{

parentNODE.rightchild = n;

return;

}

}

}

}

}

public void InOrderTraversal(NODE currentNODE)

{

if(currentNODE != null)

{

InOrderTraversal(currentNODE.leftchild);

System.out.println("\t" + currentNODE.Name + " - " + currentNODE.phoneNo);

InOrderTraversal(currentNODE.rightchild);

}

}

public String minNODE(NODE currentNODE)

{

while(currentNODE.leftchild != null)

{

currentNODE = currentNODE.leftchild;

}

return ("\t" + currentNODE.Name + " has LEAST balance of Rs." + currentNODE.Balance);

}

public String maxNODE(NODE currentNODE)

{

while(currentNODE.rightchild != null)

{

currentNODE = currentNODE.rightchild;

}

return ("\t" + currentNODE.Name + " has HIGHEST balance of Rs." + currentNODE.Balance);

}

public static void ListMenu()

{

System.out.println("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("A. Add a customer: ");

System.out.println("B. Display Minimum and Maximum Balance accounts:");

System.out.println("C. List all the customers: ");

System.out.println("D. Range of balance:");

System.out.println("E. Quit the program");

System.out.println("\nEnter an option: ");

}

public static void PrintRange(NODE root, int k1, int k2)

{

if (root == null) {

return;

}

if (k1 < root.Balance) {

PrintRange(root.leftchild, k1, k2);

}

if (k1 <= root.Balance && k2 >= root.Balance) {

System.out.println("\t" + root.phoneNo);

}

if (k2 > root.Balance) {

PrintRange(root.rightchild, k1, k2);

}

}

public static void main(String[] args) {

ADMIN tree = new ADMIN();

Scanner input = new Scanner(System.in);

long phoneNo;

int Balance;

String Name;

System.out.print("Enter number of people: ");

int noOfCustomer = input.nextInt();

for (int i = 0; i < noOfCustomer; i++)

{

System.out.println("\nCustomer " + (1+i) + " record: ");

System.out.print("\tEnter phone number of customer " + (1+i) + ": ");

phoneNo = input.nextLong();

while(phoneNo == 0 || phoneNo < 0)

{

System.out.print("\t -- Phone Number must be graeter than 0");

System.out.print("\n\tEnter Phone Number: ");

phoneNo = input.nextLong();

}

System.out.print("\tEnter name of customer " + (1+i) + ": ");

Name = input.next();

while(Name.isEmpty() == true)

{

System.out.print("\t Name must contain text.");

System.out.print("\n\tEnter customer name: ");

Name = input.next();

}

System.out.print("\tEnter balance: ");

Balance = input.nextInt();

while(Balance < 0)

{

System.out.print("\t -- Balance cannot be negative.");

System.out.print("\n\tEnter Account Balance: ");

Balance = input.nextInt();

}

tree.insertion(phoneNo, Balance, Name);

}

System.out.print("\n");

tree.InOrderTraversal(tree.root);//System.out.print("\nCustomer with first phoneNo: ");

System.out.println(tree.minNODE(tree.root)); //System.out.print("\nCustomer with last phoneNo: ");

System.out.println(tree.maxNODE(tree.root));

ListMenu();

String option = input.next();

option = option.toUpperCase();

boolean isFinished = false;

while(!isFinished)

{

try {

switch (option)

{

case "A":

System.out.print("\tEnter Phone Number: ");

phoneNo = input.nextLong();

while(phoneNo == 0 || phoneNo < 0)

{

System.out.print("\t -- Phone Number must be graeter than 0");

System.out.print("\n\tEnter Phone Number: ");

phoneNo = input.nextLong();

}

System.out.print("\tEnter cutomer name: ");

Name = input.next();

while(Name.isEmpty() == true)

{

System.out.print("\t Name must contain text.");

System.out.print("\n\tEnter customer name: ");

Name = input.next();

}

System.out.print("\tEnter Account Balance: ");

Balance = input.nextInt();

while(Balance < 0)

{

System.out.print("\t -- Balance cannot be negative.");

System.out.print("\n\tEnter Account Balance: ");

Balance = input.nextInt();

}

tree.insertion(phoneNo, Balance, Name);

ListMenu();

option = input.next().toUpperCase();

break;

case "B":

System.out.println(tree.minNODE(tree.root));

System.out.println(tree.maxNODE(tree.root));

ListMenu();

option = input.next().toUpperCase();

break;

case "C":

tree.InOrderTraversal(tree.root);

ListMenu();

option = input.next().toUpperCase();

break;

case "D":

System.out.print("\tEnter range start: ");

int k1 = input.nextInt();

System.out.print("\tEnter range end: ");

int k2 = input.nextInt();

System.out.print("\n");

PrintRange(root, k1, k2);

ListMenu();

option = input.next().toUpperCase();

break;

case "E":

isFinished = true;

break;

default:

System.out.print("\n\tInvalid input. Enter a new input\n");

option = input.next().toUpperCase();

break;

}

}

catch(Exception e)

{

System.out.println("Error Occured.");

break;

}

}

}

}