City MSLL

Ahmed alnemer

201837500

SLLNode class:

public class SLLNode<T> {  
 public T info;  
 public SLLNode<T> next;  
 public SLLNode() {  
 this(null,null);  
 }  
 public SLLNode(T el) {  
 this(el,null);  
 }  
 public SLLNode(T el, SLLNode<T> ptr) {  
 info = el;  
 next = ptr;  
 }  
}

MSLLNode class:

public class MSLLNode<T> {  
 public T info;  
 public SLL<T> list;  
 public MSLLNode<T> next;  
  
 public MSLLNode() {  
 info = null;  
 list = null;  
 next = null;  
 }  
 public MSLLNode(T el) {  
 this(el,null);  
 }  
 public MSLLNode(SLL<T> list) {  
 this(list,null);  
 }  
 public MSLLNode(T el, MSLLNode<T> ptr) {  
 info = el;  
 next = ptr;  
 }  
 public MSLLNode(SLL<T> list, MSLLNode<T> ptr) {  
 this.list = list;  
 next = ptr;  
 }  
 public MSLLNode(T el, SLL<T> list, MSLLNode<T> ptr) {  
 info = el;  
 this.list = list;  
 next = ptr;  
 }  
}

MSLL class

public class MSLL<T> {  
 protected MSLLNode<T> head, tail;  
  
 public MSLL() {  
 head = tail = null;  
 }  
 public boolean isEmpty() {  
 return head == null;  
 }  
 public void addToMSLLHead(String s1) throws Exception {  
 if (isInMSLList(s1))  
 throw new Exception("element already exist");  
 else {  
 head = new MSLLNode(s1, new SLL(), head);  
 if (tail == null)  
 tail = head;  
 }  
 }  
 public void addToMSLLTail(String s1) throws Exception {  
 if (isInMSLList(s1))  
 throw new Exception("element already exist");  
 else {  
 if (!isEmpty()) {  
 tail.next = new MSLLNode(s1, new SLL(), head);  
 tail = tail.next;  
 } else head = tail = new MSLLNode(s1, new SLL(), head);  
 }  
 }  
  
 public boolean isInMSLList(String s1) {  
 if (isEmpty())  
 return false;  
 else {  
 MSLLNode<T> tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s1); tmp = tmp.next) ;  
 return tmp != null;  
 }  
 }  
  
 public void addCityToSublistAtRear(City cityObject) throws Exception {  
  
 char a = cityObject.getName().charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
 if (!isInMSLList(s)) {  
 addToMSLLTail(s);  
 }  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next);  
  
 if (tmp.list.isInList(cityObject))  
 throw new Exception("city already exist");  
 else  
 tmp.list.addToTail(cityObject);  
  
  
  
  
 }  
  
 public void addToCitySublistList(City c1, int position) throws Exception {  
  
 char a = c1.getName().charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
 if (!isInMSLList(s)) {  
 addToMSLLTail(s);  
 }  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next);  
  
 if (tmp.list.isInList(c1))  
 throw new Exception("city already exist");  
 else  
 tmp.list.incertAtPosition(c1,position);  
  
 }  
  
  
  
  
 public boolean cityIsInSubList(City c1){  
 if (isEmpty())  
 return false;  
 else {  
 char a = c1.getName().charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next) ;  
  
  
 if (tmp != null)  
 return tmp.list.isInList(c1);  
 else  
 return false;  
 }  
 }  
  
 public void deleteCityFromSublist(City c1) throws Exception {  
  
 if (!cityIsInSubList(c1))  
 throw new Exception("city already exist");  
  
 char a = c1.getName().charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next);  
 tmp.list.delete(c1);  
  
 }  
  
 public void deleteCitySublist(String str) throws Exception {  
 if (!isInMSLList(str))  
 throw new Exception("city is not in sublist");  
 else {  
 char a = str.charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next) ;  
 tmp.list = null;  
 }  
 }  
  
 public void makeCitySublistEmpty(String str) throws Exception {  
 if (!isInMSLList(str))  
 throw new Exception("elemnt is not in MSLL");  
 else {  
 char a = str.charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next) ;  
 tmp.list.clear();  
 }  
 }  
  
 public void displayCitySublist(String s1) throws Exception {  
 if (!isInMSLList(s1))  
 throw new Exception("elemnt is not in MSLL");  
 else {  
 char a = s1.charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next) ;  
 tmp.list.printAll();  
 }  
  
  
 }  
  
  
 public void deleteFromMSLL(String s1){  
 if (!isEmpty())  
 if (head == tail && s1.equals(head.info)) // if only one  
 head = tail = null; // node on the list;  
 else if (s1.equals(head.info)) // if more than one node on the list;  
 head = head.next; // and el is in the head node;  
 else { // if more than one node in the list  
 MSLLNode<T> pred, tmp;// and el is in a nonhead node;  
 for (pred = head, tmp = head.next;  
 tmp != null && !tmp.info.equals(s1);  
 pred = pred.next, tmp = tmp.next);  
 if (tmp != null) { // if el was found;  
 pred.next = tmp.next;  
 if (tmp == tail) // if el is in the last node;  
 tail = pred;  
 }  
 }  
 }  
  
  
 public double getDistance(String city1, String city2) throws Exception {  
 City c1 = new City(city1);  
 City c2 = new City(city2);  
  
 if (!cityIsInSubList(c1) || !cityIsInSubList(c2)) {  
 throw new Exception("City is not in sublist");  
 } else {  
  
 char a = city1.charAt(0);  
 String s = "" + Character.*toUpperCase*(a);  
  
 MSLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next) ;  
  
 SLLNode city;  
 for (city = tmp.list.head; city != null; city = city.next) {  
 if (c1.compareTo((City) city.info) == 0)  
 c1 = (City) city.info;  
 }  
  
  
 a = city2.charAt(0);  
 s = "" + Character.*toUpperCase*(a);  
  
  
 for (tmp = head; tmp != null && !tmp.info.equals(s); tmp = tmp.next) ;  
  
  
 for (city = tmp.list.head; city != null; city = city.next) {  
 if (c2.compareTo((City) city.info) == 0)  
 c2 = (City) city.info;  
 }  
 double distance;  
 double φ1 = c1.decimalLatitude;  
 double φ2 = c2.decimalLatitude;  
 double λ1 = c1.decimalLongitude;  
 double λ2 = c2.decimalLongitude ;  
  
 double latitudeD = Math.*toRadians*(φ2 - φ1);  
 double longitudeD = Math.*toRadians*(λ2 - λ1);  
  
 φ1 = Math.*toRadians*(c1.decimalLatitude);  
 φ2 = Math.*toRadians*(c2.decimalLatitude);  
  
 double d = Math.*pow*(Math.*sin*( latitudeD / 2), 2) + Math.*cos*(φ1) \* Math.*cos*(φ2) \* Math.*pow*(Math.*sin*( longitudeD / 2), 2);  
 distance = 2 \* City.*r* \* Math.*asin*(Math.*sqrt*(d));  
 return distance;  
 }  
 }  
  
 public int length() {  
 if (isEmpty()) {  
 return 0;  
 } else {  
 int length = 0;  
 for (MSLLNode tmp = head; tmp != null; tmp = tmp.next) {  
 length++;  
 }  
 return length;  
 }  
 }  
}

SLL class:

public class SLL<T> {  
 protected SLLNode<T> head, tail;  
 public SLL() {  
 head = tail = null;  
 }  
 public boolean isEmpty() {  
 return head == null;  
 }  
 public void addToHead(T el) {  
 head = new SLLNode<T>(el,head);  
 if (tail == null)  
 tail = head;  
 }  
  
 public void clear() {  
 head = tail = null;  
 }  
  
 public void addToTail(T el) {  
 if (!isEmpty()) {  
 tail.next = new SLLNode<T>(el);  
 tail = tail.next;  
 }  
 else head = tail = new SLLNode<T>(el);  
 }  
  
 public void incertAtPosition(T el, int i) throws IllegalAccessException {  
 SLLNode<T> tmp;  
 SLLNode<T> newNode = new SLLNode<T>(el);  
  
 if (isInList(el))  
 throw new IllegalAccessException("City already exist");  
  
 if (i <= 0 || i > length())  
 throw new IllegalAccessException("invaled position ");  
  
 if (i ==1)  
 addToHead((T) el);  
 else {  
 int p = 1;  
 for (tmp = head; tmp != null && p != i-1 ; tmp = tmp.next, p++);  
 newNode.next = tmp.next;  
 tmp.next = newNode;  
 }  
  
 }  
  
 public T deleteFromHead() { // delete the head and return its info;  
 if (isEmpty())  
 return null;  
 T el = head.info;  
 if (head == tail) // if only one node on the list;  
 head = tail = null;  
 else head = head.next;  
 return el;  
 }  
 public T deleteFromTail() { // delete the tail and return its info;  
 if (isEmpty())  
 return null;  
 T el = tail.info;  
 if (head == tail) // if only one node in the list;  
 head = tail = null;  
 else { // if more than one node in the list,  
 SLLNode<T> tmp; // find the predecessor of tail;  
 for (tmp = head; tmp.next != tail; tmp = tmp.next);  
 tail = tmp; // the predecessor of tail becomes tail;  
 tail.next = null;  
 }  
 return el;  
 }  
  
 public void delete(T el) { // delete the node with an element el;  
 if (!isEmpty())  
 if (head == tail && el.equals(head.info)) // if only one  
 head = tail = null; // node on the list;  
 else if (el.equals(head.info)) // if more than one node on the list;  
 head = head.next; // and el is in the head node;  
 else { // if more than one node in the list  
 SLLNode<T> pred, tmp;// and el is in a nonhead node;  
 for (pred = head, tmp = head.next;  
 tmp != null && !tmp.info.equals(el);  
 pred = pred.next, tmp = tmp.next);  
 if (tmp != null) { // if el was found;  
 pred.next = tmp.next;  
 if (tmp == tail) // if el is in the last node;  
 tail = pred;  
 }  
 }  
 }  
  
 public void delete(City el) { // delete the node with an element el;  
 if (!isEmpty())  
 if (head == tail && el.equals((City) head.info)) // if only one  
 head = tail = null; // node on the list;  
 else if (el.equals((City) head.info)) // if more than one node on the list;  
 head = head.next; // and el is in the head node;  
 else { // if more than one node in the list  
 SLLNode<T> pred, tmp;// and el is in a nonhead node;  
 for (pred = head, tmp = head.next;  
 tmp != null && !((City) tmp.info).equals(el);  
 pred = pred.next, tmp = tmp.next)  
 ;  
 if (tmp != null) { // if el was found;  
 pred.next = tmp.next;  
 if (tmp == tail) // if el is in the last node;  
 tail = pred;  
 }  
 }  
 }  
  
 public void printAll() {  
 for (SLLNode<T> tmp = head; tmp != null; tmp = tmp.next)  
 System.*out*.print(tmp.info.toString() + "\n\n");  
 }  
  
 public boolean isInList(T el) {  
 SLLNode tmp;  
 for (tmp = head; tmp != null && !tmp.info.equals(el); tmp = tmp.next);  
 return tmp != null;  
 }  
 public boolean isInList(City el) {  
 SLLNode tmp;  
 for (tmp = head; tmp != null && !((City)tmp.info).equals(el); tmp = tmp.next);  
 return tmp != null;  
 }  
  
 public int length(){  
 int length=0;  
 if (isEmpty())  
 return 0;  
 else  
 for (SLLNode<T> tmp = head; tmp != null; tmp = tmp.next) {  
 length++;  
 }  
 return length;  
 }  
  
  
}

City class

public class City {  
 String name;  
 double decimalLatitude;  
 double decimalLongitude;  
 static final double *r* = 6372.8;  
  
 public City(){  
  
 }  
 public City(String name){  
 this.name = name;  
 }  
  
 public City(String name, double decimalLatitude, double decimalLongitude){  
 this.name = name;  
 this.decimalLatitude = decimalLatitude;  
 this.decimalLongitude = decimalLongitude;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getDecimalLatitude() {  
 return decimalLatitude;  
 }  
  
 public double getDecimalLongitude() {  
 return decimalLongitude;  
 }  
  
 public int compareTo(City o){  
 return name.compareTo(o.name);  
 }  
  
 @Override  
 public String toString() {  
 return  
 "name=" + name + '\n' +  
 "decimalLatitude=" + decimalLatitude +  
 "\nndecimalLongitude=" + decimalLongitude;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public void setDecimalLongitude(double decimalLongitude) {  
 this.decimalLongitude = decimalLongitude;  
 }  
  
 public void setDecimalLatitude(double decimalLatitude) {  
 this.decimalLatitude = decimalLatitude;  
 }  
 public boolean equals(City c){  
 return name.equals(c.getName());  
 }  
  
  
}

driver class

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class Driver {  
  
  
 public static void main(String[] args) throws Exception {  
  
  
  
  
  
   
  
 Scanner kb = new Scanner(System.*in*);  
 MSLL<City> citys = new MSLL<>();  
 String city ,city2;  
 double decimalLatitude;  
 double decimalLongitude;  
 int choice = 0;  
 int p;  
 City c;  
  
  
  
 do {  
 try {  
 System.*out*.println();  
 System.*out*.println("1.\tAddCityToSublistAtRear\n" +  
 "2.\tAddCityToSublistAtPosition\n" +  
 "3.\tSearchForCity\n" +  
 "4.\tDeleteCityFromCitySublist\n" +  
 "5.\tDeleteCitySublist\n" +  
 "6.\tMakeCitySublistEmpty\n" +  
 "7.\tDisplayCitySublist\n" +  
 "8.\taddToMSSLAtHead\n" +  
 "9.\tAddToMSSLAtTail\n" +  
 "10.\tSearchMSLList\n" +  
 "11.\tgetDistance\n" +  
 "12.\tExit\n");  
  
 choice = kb.nextInt();  
  
  
 if (choice > 12 || choice <= 0)  
 throw new IllegalAccessException("wrong menu choice");  
  
  
 switch (choice) {  
  
  
 case 1:  
 System.*out*.println("enter city name: ");  
 city = kb.next();  
 System.*out*.println("enter city decimal Latitude: ");  
 decimalLatitude = kb.nextDouble();  
 System.*out*.println("enter city decimal Longitude: ");  
 decimalLongitude = kb.nextDouble();  
 c = new City(city, decimalLatitude, decimalLongitude);  
 citys.addCityToSublistAtRear(c);  
 System.*out*.println("city is added to sublist ");  
 break;  
  
  
 case 2:  
 System.*out*.println("enter city name: ");  
 city = kb.next();  
 System.*out*.println("enter city decimal Latitude: ");  
 decimalLatitude = kb.nextDouble();  
 System.*out*.println("enter city decimal Longitude: ");  
 decimalLongitude = kb.nextDouble();  
 c = new City(city, decimalLatitude, decimalLongitude);  
 System.*out*.println("enter wanted position: ");  
 p = kb.nextInt();  
 citys.addToCitySublistList(c, p);  
 System.*out*.println("city is added to sublist in position "+p);  
  
  
 break;  
  
 case 3:  
 System.*out*.println("enter city name: ");  
 city = kb.next();  
 c = new City(city);  
  
 if (citys.cityIsInSubList(c))  
 System.*out*.println("city is in a subList");  
 else  
 System.*out*.println("city is not in a subList");  
  
  
 break;  
  
 case 4:  
 System.*out*.println("enter city name: ");  
 city = kb.next();  
 c = new City(city);  
 citys.deleteCityFromSublist(c);  
 System.*out*.println("city is deleted from subList");  
  
 break;  
  
 case 5:  
 System.*out*.println("enter capital character: ");  
 city = kb.next();  
  
 while (Character.*isLowerCase*(city.charAt(0)) || city.length() > 1) {  
 System.*out*.println("invaled input, please enter capital character: ");  
 city = kb.next();  
  
 }  
 citys.deleteCitySublist(city);  
 System.*out*.println("subList is deleted");  
  
 break;  
  
 case 6:  
 System.*out*.println("enter capital character: ");  
 city = kb.next();  
  
 while (Character.*isLowerCase*(city.charAt(0)) || city.length() > 1) {  
 System.*out*.println("invaled input, please enter capital character: ");  
 city = kb.next();  
  
 }  
 citys.makeCitySublistEmpty(city);  
 System.*out*.println("subList is empty");  
  
 break;  
  
 case 7:  
 System.*out*.println("enter capital character: ");  
 city = kb.next();  
  
 while (Character.*isLowerCase*(city.charAt(0)) || city.length() > 1) {  
 System.*out*.println("invaled input, please enter capital character: ");  
 city = kb.next();  
  
 }  
 citys.displayCitySublist(city);  
  
  
 break;  
  
 case 8:  
 System.*out*.println("enter capital character: ");  
 city = kb.next();  
  
 while (Character.*isLowerCase*(city.charAt(0)) || city.length() > 1) {  
 System.*out*.println("invaled input, please enter capital character: ");  
 city = kb.next();  
  
 }  
 citys.addToMSLLHead(city);  
 System.*out*.println("element is added to MSLL head");  
  
 break;  
  
 case 9:  
 System.*out*.println("enter capital character: ");  
 city = kb.next();  
  
 while (Character.*isLowerCase*(city.charAt(0)) || city.length() > 1) {  
 System.*out*.println("invaled input, please enter capital character: ");  
 city = kb.next();  
  
 }  
 citys.addToMSLLTail(city);  
 System.*out*.println("element is added to MSLL tail");  
 break;  
  
 case 10:  
 System.*out*.println("enter capital character: ");  
 city = kb.next();  
  
 while (Character.*isLowerCase*(city.charAt(0)) || city.length() > 1) {  
 System.*out*.println("invaled input, please enter capital character: ");  
 city = kb.next();  
  
 }  
 if (citys.isInMSLList(city)) {  
 System.*out*.println("elemnt is in MSLL list ");  
 } else  
 System.*out*.println("elemnt is in not MSLL list ");  
  
  
 break;  
  
 case 11:  
 System.*out*.println("enter city 1: ");  
 city = kb.next();  
 System.*out*.println("enter city 2: ");  
 city2 = kb.next();  
 double d = citys.getDistance(city, city2);  
 System.*out*.println("distance between the two citys is : " + d);  
  
  
 break;  
  
 case 12:  
 kb.close();  
 System.*exit*(0);  
  
 break;  
 }  
  
 } catch (IllegalArgumentException e) {  
 System.*out*.println(e.getMessage());  
 }catch (InputMismatchException e){  
 System.*out*.println("wrong input \n ");  
 kb.nextLine();  
 }  
  
  
 } while (choice != 12);  
  
  
  
 }  
 }

output screen shots:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated