* Create Position Interface
* Create Positional List interface
* Create Positional Linked List Using Linked List structure (Nodes)
* Implement Basic Methods of Positional Linked List
* addBefore(Position<E> p ,E e)
* addAfter(Position<E> p ,E e)
* remove(Position<E> p)
* Implement Iterator and Iterable pattern design in Positional Linked Lists

إنشاء واجهة Position:

public interface Position<E> {

E element(); // إرجاع العنصر الموجود في الموقف

}

public interface PositionalList<E> {

Position<E> addFirst(E e); // إضافة عنصر في البداية

Position<E> addLast(E e); // إضافة عنصر في النهاية

Position<E> addBefore(Position<E> p, E e); // إضافة عنصر قبل موقف معين

Position<E> addAfter(Position<E> p, E e); // إضافة عنصر بعد موقف معين

E remove(Position<E> p); // إزالة عنصر من موقف معين

Position<E> first(); // إرجاع أول موقف

Position<E> last(); // إرجاع آخر موقف

Position<E> before(Position<E> p); // إرجاع الموقف السابق

Position<E> after(Position<E> p); // إرجاع الموقف التالي

boolean isEmpty(); // التحقق مما إذا كانت القائمة فارغة

int size(); // إرجاع عدد العناصر

}

إنشاء واجهة PositionalList:

public class Node<E> implements Position<E> {

private E element;

private Node<E> prev; // العقدة السابقة

private Node<E> next; // العقدة التالية

public Node(E e, Node<E> p, Node<E> n) {

element = e;

prev = p;

next = n;

}

public E element() {

return element;

}

public Node<E> getPrev() {

return prev;

}

public Node<E> getNext() {

return next;

}

public void setPrev(Node<E> p) {

prev = p;

}

public void setNext(Node<E> n) {

next = n;

}

}

إنشاء قائمة مرتبطة موضعية باستخدام بنية قائمة مرتبطة (Nodes):

public class Node<E> implements Position<E> {

private E element;

private Node<E> prev; // العقدة السابقة

private Node<E> next; // العقدة التالية

public Node(E e, Node<E> p, Node<E> n) {

element = e;

prev = p;

next = n;

}

public E element() {

return element;

}

public Node<E> getPrev() {

return prev;

}

public Node<E> getNext() {

return next;

}

public void setPrev(Node<E> p) {

prev = p;

}

public void setNext(Node<E> n) {

next = n;

}

}

تنفيذ الأساليب الأساسية لقائمة مرتبطة موضعية:

public class LinkedPositionalList<E> implements PositionalList<E> {

private Node<E> header; // رأس القائمة

private Node<E> trailer; // ذيل القائمة

private int size = 0; // عدد العناصر

public LinkedPositionalList() {

header = new Node<>(null, null, null);

trailer = new Node<>(null, header, null);

header.setNext(trailer);

}

@Override

public Position<E> addBefore(Position<E> p, E e) {

Node<E> node = validate(p);

Node<E> newNode = new Node<>(e, node.getPrev(), node);

node.getPrev().setNext(newNode);

node.setPrev(newNode);

size++;

return newNode;

}

@Override

public Position<E> addAfter(Position<E> p, E e) {

Node<E> node = validate(p);

Node<E> newNode = new Node<>(e, node, node.getNext());

node.getNext().setPrev(newNode);

node.setNext(newNode);

size++;

return newNode;

}

@Override

public E remove(Position<E> p) {

Node<E> node = validate(p);

node.getPrev().setNext(node.getNext());

node.getNext().setPrev(node.getPrev());

size--;

return node.element();

}

private Node<E> validate(Position<E> p) {

if (!(p instanceof Node)) throw new IllegalArgumentException("Invalid position");

Node<E> node = (Node<E>) p;

if (node.getNext() == null) throw new IllegalArgumentException("Position is no longer in the list");

return node;

}

// تنفيذ باقي الأساليب مثل first(), last(), before(), after(), isEmpty(), size()

}

:5. تنفيذ نمط التصميم Iterator وIterable في القوائم الموضعية

public class LinkedPositionalList<E> implements Iterable<E> {

// ... (كود القائمة المرتبطة)

@Override

public Iterator<E> iterator() {

return new Iterator<E>() {

private Position<E> current = first();

@Override

public boolean hasNext() {

return current != null;

}

@Override

public E next() {

E element = current.element();

current = after(current); // الانتقال إلى العنصر التالي

return element;

}

};

}

}

------------------------------------------------------------------------

Homework

* Implement the ArrayList Data structure as it is described in chapter 7.

import java.util.Arrays;

public class ArrayList<E> {

private E[] array;

private int size = 0;

public ArrayList() {

array = (E[]) new Object[10]; // مصفوفة ابتدائية

}

public void add(E e) {

if (size == array.length) {

resize();

}

array[size++] = e;

}

private void resize() {

array = Arrays.copyOf(array, array.length \* 2);

}

public E get(int index) {

if (index < 0 || index >= size) throw new IndexOutOfBoundsException();

return array[index];

}

// إضافة المزيد من الأساليب مثل remove(), size(), isEmpty()

}

-----------------------------------------------------------------------------

* Implement the iterator idea in your ArrayList.

public class ArrayList<E> implements Iterable<E> {

// ... (كود ArrayList)

@Override

public Iterator<E> iterator() {

return new Iterator<E>() {

private int currentIndex = 0;

@Override

public boolean hasNext() {

return currentIndex < size;

}

@Override

public E next() {

if (!hasNext()) throw new NoSuchElementException();

return array[currentIndex++];

}

};

}

}