## Topics

1. Create Position Interface
2. Create Positional List interface
3. Create Positional Linked List Using Linked List structure (Nodes)
4. Implement Basic Methods of Positional Linked List

* addBefore(Position<E> p ,E e)
* addAfter(Position<E> p ,E e)
* remove(Position<E> p)

1. Implement Iterator and Iterable pattern design in Positional Linked Lists

public interface Position<E> {

E getElement();

}

public interface PositionalList<E> {

Position<E> addBefore(Position<E> p, E e);

Position<E> addAfter(Position<E> p, E e);

Position<E> remove(Position<E> p);

}

import java.util.Iterator;

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

private static class Node<E> implements Position<E> {

private E element;

private Node<E> prev, next;

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

element = e;

prev = p;

next = n;

}

public E getElement() {

return element;

}

public Node<E> getPrev() {

return prev;

}

public void setPrev(Node<E> prev) {

this.prev = prev;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

}

private Node<E> header, trailer;

private int size = 0;

public LinkedPositionalList() {

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

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

header.setNext(trailer);

}

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if (node.getNext() == null) throw new IllegalArgumentException("Position no longer valid");

return node;

}

@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 Position<E> remove(Position<E> p) {

Node<E> node = validate(p);

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

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

size--;

return node;

}

@Override

public Iterator<E> iterator() {

return new Iterator<E>() {

private Node<E> current = header.getNext();

@Override

public boolean hasNext() {

return current != trailer;

}

@Override

public E next() {

E element = current.getElement();

current = current.getNext();

return element;

}

};

}

}

import java.util.Iterator;

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

private E[] data;

private int size = 0;

private static final int DEFAULT\_CAPACITY = 10;

public CustomArrayList() {

data = (E[]) new Object[DEFAULT\_CAPACITY];

}

public void add(E e) {

if (size == data.length) resize(2 \* data.length);

data[size++] = e;

}

public E remove(int index) {

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

E element = data[index];

for (int i = index; i < size - 1; i++) {

data[i] = data[i + 1];

}

data[--size] = null;

return element;

}

private void resize(int capacity) {

E[] newData = (E[]) new Object[capacity];

System.arraycopy(data, 0, newData, 0, size);

data = newData;

}

@Override

public Iterator<E> iterator() {

return new Iterator<E>() {

private int index = 0;

@Override

public boolean hasNext() {

return index < size;

}

@Override

public E next() {

return data[index++];

}

};

}

}

## Homework

1. Implement the ArrayList Data structure as it is described in chapter 7.
2. Implement the iterator idea in your ArrayList.

import java.util.Iterator;

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

private E[] data;

private int size = 0;

private static final int DEFAULT\_CAPACITY = 10;

// المُنشئ الافتراضي

public CustomArrayList() {

data = (E[]) new Object[DEFAULT\_CAPACITY];

}

// إضافة عنصر إلى القائمة

public void add(E e) {

if (size == data.length) resize(2 \* data.length); // توسيع المصفوفة عند الامتلاء

data[size++] = e;

}

// إزالة عنصر من القائمة حسب الفهرس

public E remove(int index) {

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

E element = data[index];

for (int i = index; i < size - 1; i++) {

data[i] = data[i + 1];

}

data[--size] = null;

return element;

}

// توسيع المصفوفة عند الحاجة

private void resize(int capacity) {

E[] newData = (E[]) new Object[capacity];

System.arraycopy(data, 0, newData, 0, size);

data = newData;

}

// الحصول على حجم القائمة

public int size() {

return size;

}

// إرجاع العنصر في موقع معين

public E get(int index) {

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

return data[index];

}

// تنفيذ Iterator للسماح بالتنقل بين عناصر القائمة باستخدام `for-each`

@Override

public Iterator<E> iterator() {

return new Iterator<E>() {

private int index = 0;

@Override

public boolean hasNext() {

return index < size;

}

@Override

public E next() {

return data[index++];

}

};

}

// اختبار القائمة

public static void main(String[] args) {

CustomArrayList<String> list = new CustomArrayList<>();

list.add("Java");

list.add("Python");

list.add("C++");

System.out.println("محتوى القائمة:");

for (String item : list) {

System.out.println(item);

}

list.remove(1);

System.out.println("\nبعد إزالة العنصر الثاني:");

for (String item : list) {

System.out.println(item);

}

}

}