## 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

## 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;

import java.util.NoSuchElementException;

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

// Instance variables

public static final int CAPACITY = 16; // Default array capacity

private E[] data; // Generic array used for storage

private int size = 0; // Current number of elements

// Constructors

public ArrayList() {

this(CAPACITY); // Constructs list with default capacity

}

public ArrayList(int capacity) {

data = (E[]) new Object[capacity]; // Safe cast; compiler may give a warning

}

// Methods

public int size() {

return size;

}

public boolean isEmpty() {

return size == 0;

}

public E get(int i) {

if (i < 0 || i >= size) {

throw new IndexOutOfBoundsException("Index out of range");

}

return data[i];

}

public void set(int i, E e) {

if (i < 0 || i >= size) {

throw new IndexOutOfBoundsException("Index out of range");

}

data[i] = e;

}

public void add(int i, E e) {

if (size == data.length) {

resize(2 \* data.length); // Double the capacity if array is full

}

if (i < 0 || i > size) {

throw new IndexOutOfBoundsException("Index out of range");

}

for (int j = size - 1; j >= i; j--) {

data[j + 1] = data[j]; // Shift elements up

}

data[i] = e;

size++;

}

public E remove(int i) {

if (i < 0 || i >= size) {

throw new IndexOutOfBoundsException("Index out of range");

}

E removedElement = data[i];

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

data[j] = data[j + 1]; // Shift elements down

}

data[size - 1] = null; // Dereference last element

size--;

return removedElement;

}

// Resize method to increase array capacity

private void resize(int capacity) {

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

for (int i = 0; i < size; i++) {

newArray[i] = data[i];

}

data = newArray;

}

// Iterator implementation

public Iterator<E> iterator() {

return new ArrayListIterator();

}

// ArrayListIterator class

private class ArrayListIterator implements Iterator<E> {

private int current = 0;

@Override

public boolean hasNext() {

return current < size;

}

@Override

public E next() {

if (!hasNext()) {

throw new NoSuchElementException("No more elements");

}

return data[current++];

}

}

// Main method for testing

public static void main(String[] args) {

ArrayList<Integer> list = new ArrayList<>();

// Adding elements

list.add(0, 10);

list.add(1, 20);

list.add(1, 15);

// Using the iterator to print elements

Iterator<Integer> iterator = list.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

// Removing an element

list.remove(1);

// Using the iterator again after removal

System.out.println("After removal:");

iterator = list.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

} }}