**The OOP Language Programming**

**Course Exercise**

**Name: AHMED MD JUBAIR**

**Student ID: 1911521211**

**Class: 19lq CST**

**Date: 2022-01-05**

**Course exercise for Object oriented programming**

3. Re-Implementing a doubly-Linked lists.

Given:

package list;

abstract public class ListNode{

protected Object item;

protected List myList;

public boolean isValidNode(){

return myList != null; }

//next() returns the node following this node. If this node is invalid, throws an exception.

public abstract ListNode next() throws InvalidNodeException;

//prev() returns the node preceding this node

public abstract ListNode prev() throws InvalidNodeException;

//insertBefore() inserts a node holding this "item" immediately preceding this node. If this node is invalid, throws an exception.

public abstract void insertBefore(Object item) throws InvalidNodeException;

// insertAfter() inserts a node holding this "item" immediately following this node. If this node is invalid, throws an exception.

public abstract void insertAfter(Object item) throws InvalidNodeException;

//remove() removes this node from its list. If this node is invalid, throws an exception.

public abstract void remove() throws InvalidNodeException;

}

package list;

abstract public class List{

protected int size;

//insertFront() inserts a node holding this "item" at the front of this list.

public abstract void insertFront(Object item);

//insertBack() inserts a node holding this "item" at the back of this list.

public abstract void insertBack(Object item);

public abstract ListNode front();

public abstract ListNode back();

public abstract String toString();

}

package list;

public class InvalidNodeException extends Exception {

protected InvalidNodeException() { super(); }

protected InvalidNodeException(String s) { super(s); }

}

package list;

public class DListNode extends ListNode{

protected DListNode prev;

protected DListNode next;

DListNode(Object o, DList dl, DListNode p, DListNode n){

//Your code }

public ListNode next() throws InvalidNodeException{

if(!isValidNode()){throw new InvalidNodeException("next() is called on invalid node");}

;//Your code

}

public ListNode prev() throws InvalidNodeException{

if(!isValidNode()){throw new InvalidNodeException("prev() is called on invalid node");}

;//Your code

}

public void insertBefore(Object item) throws InvalidNodeException{

if(!isValidNode()){throw new InvalidNodeException("insertBefore() is called on invalid node");}

//Your code

}

public void insertAfter(Object item) throws InvalidNodeException{

if(!isValidNode()){throw new InvalidNodeException("insertAfter() is called on invalid node");}

//Your code

}

public void remove() throws InvalidNodeException{

if(!isValidNode()){throw new InvalidNodeException("remove() is called on invalid node");}

//Your code

prev = null; next = null; myList = null;

}

}

package list;

public class DList extends List{

protected DListNode head;

protected DListNode newNode(Object item, DList list, DListNode prev, DListNode next){

return new DListNode(item, list, prev, next);

}

public DList(){ //Your code

}

public void insertFront(Object item){ //Your code

}

public void insertBack(Object item){ //Your code

}

public ListNode front(){ return head.next; }

public ListNode back(){ return head.prev; }

public String toString(){

//Your code

}

}

• finish coding the 6 methods defined in the DListNode class, and explain your design.

DListNode(Object o, DList dl, DListNode p, DListNode n){

item = o;

myList = dl;

prev = p;

next = n;

}

public ListNode next() throws InvalidNodeException{

if(!isValidNode()){

throw new InvalidNodeException("next() is called on invalid node");}

return next;

}

public ListNode prev() throws InvalidNodeException{

if(!isValidNode()){

throw new InvalidNodeException("prev() is called on invalid node");}

return prev;

}

public void insertBefore(Object item) throws InvalidNodeException{

if(!isValidNode()){

throw new InvalidNodeException("insertBefore() is called on invalid node");}

DListNode insBefore\_Node = ((DList)this.myList).newNode(item, (DList)this.myList, this.prev, this);

this.prev.next = insBefore\_Node;

this.prev = insBefore\_Node;

this.myList.size ++;

}

public void insertAfter(Object item) throws InvalidNodeException{

if(!isValidNode()){

throw new InvalidNodeException("insertAfter() is called on invalid node");}

DListNode insAfter\_Node = ((DList)this.myList).newNode(item, (DList)this.myList, this, this.next);

this.next.prev = insAfter\_Node;

this.next = insAfter\_Node;

this.myList.size ++;

}

public void remove() throws InvalidNodeException{

if(!isValidNode()){

throw new InvalidNodeException("remove() is called on invalid node");}

this.prev.next = this.next;

this.next.prev = this.prev;

this.myList.size --;

prev = null; next = null; myList = null;

}

• finish coding the 4 methods defined in the DList class, and explain your design.

public DList(){

head = newNode(null, null, null, null);

head.next = head;

head.prev = head;

head.myList = this;

}

public void insertFront(Object item){

DListNode node = newNode(item, this, head, head.next);

head.next.prev = node;

head.next = node;

size ++;

}

public void insertBack(Object item){

DListNode node = newNode(item, this, head.prev, head);

head.prev.next = node;

head.prev = node;

size ++;

}

public String toString(){

String result = "[ ";

DListNode current = head.next;

while (current != head) {

result = result + current.item + " ";

current = current.next;

}

result = result + "]";

return result;

}

• Test your DList implementation using the following program. It should output the same as those in the comments.

import list.DList;

import list.DListNode;

import list.InvalidNodeException;

class Test2{

public static void main(String[] args) throws InvalidNodeException {

DList dl = new DList();

dl.insertFront(3); dl.insertFront(2); dl.insertFront(1);

dl.insertBack(4); dl.insertBack(5); dl.insertBack(6);

((DListNode)(dl.back())).insertBefore(5.5);

((DListNode)(dl.front())).insertAfter(1.5);

System.out.println(dl);//1,1.5, 2,3,4,5,5.5, 6

((DListNode)(dl.front())).remove();

System.out.println(dl);//1.5, 2,3,4,5,5.5, 6

}

}

Run your program and take a screen shot and paste it here.

