Positional Lists

Outline and Required Reading:

- Positional Lists (§ 7.3)

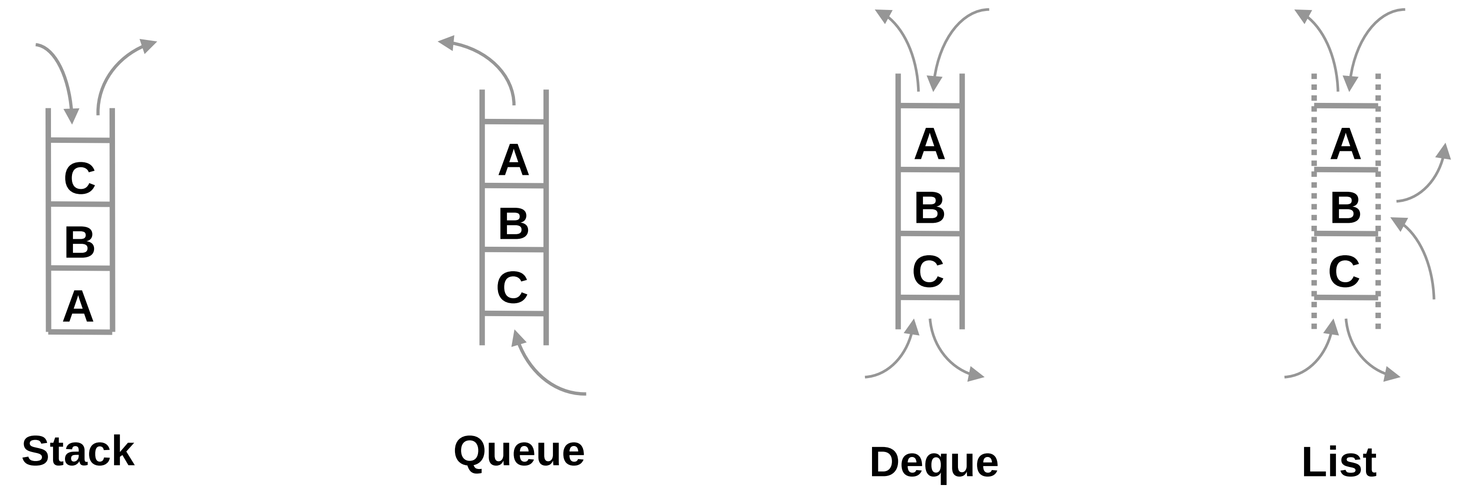
CSE 2011, Winter 2017
Instructor: N. Vlajic

List ADT

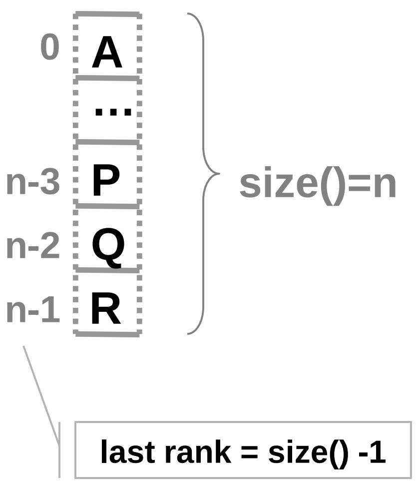
Lists – **'linear container' which allows direct access to <u>any of its elements</u>**

- **items can be accessed either through rank / index or position relative to the position of other items in the list**

Stacks, Queues, Deques – **restricted lists with methods for accessing, inserting, and removing <u>only</u> the first and/or last element**



ArrayList ADT




```
public interface ArrayList<E> {
        int size();
         /* return the # of objects in this list */
         boolean isEmpty();
         /* return true if the list is empty */

          E get(int k) throws IndexOutOfBoundsException;
          /* return the element at rank k without removing it*/
          /* error if k<0 or k≥size()=n – current # of elements */
        E set(int k, E e) throws IndexOutOfBounds…;
         /* replace with e elem. at rank k; return replaced element */
         /* error if k<0 or k≥size()=n – current # of elements */
          void add(int k, E e) throws IndexOutOfBounds…;
          /* insert a new element e into list at rank k */
          /* error if k<0 or k>size()=n – current # of elements */
          /* rank of all subsequent elements will increase! */
          E remove(int k) throws IndexOutOfBounds…;
          /* remove and return the element at rank k */
          /* error if k<0 or k≥size()=n – current # of elements */
           /* rank of all subsequent elements will decrease! */
}
```

ArrayList ADT (cont.)

Run Times in Array Implementation

Method	Time
size	O(1)
isEmpty	O(1)
get	O(1)
set	O(1)
add	O(n)
remove	O(n)

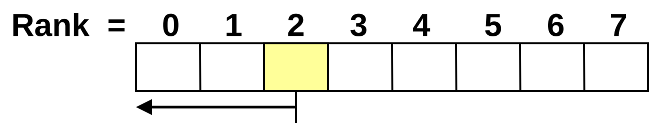
Run Times in Linked List Implementation

Method	Time
size	O(1)
isEmpty	O(1)
get	O(n)
set	O(n)
add	O(n)
remove	O(n)

Array List vs. Positional Lists

Array List – **ADT that employs "sequential allocation"**

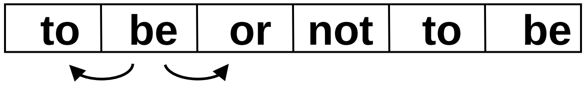
- **elements are identified by their rank / index** (sequence #)
- **no notion of spatial relation among elements,** except through rank



2 elements "before", but their identity is not known

Positional List – **ADT that employs "position allocation"**

- **each position can be used to refer to next / preceding element**
- **each position can be accessed through one of its neighbors**



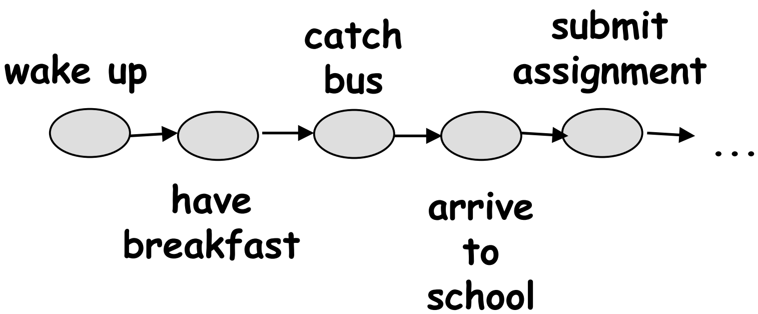
"be" is **after** "to", and **before** "or"

Positional List Application – **whenever it is required that insert / remove run in O(1) time – e.g. Text Editor: insert/remove letters at a cursor**

Array List vs. Positional Lists (cont.)

Example [Array vs. Positional List Application – building database]

Group	Name	Login
	Bui, Natalie	NB1511
	Brock, Jayden	JB1511
	Burt, Jasmine	JB1511
	Dibble, Jack	JD1511
	Dixon, Jesse	JD1511
	Dye, Frank	FD1511



Array List example: database of unrelated items - e.g. list of students organized according to their student numbers. The neighbors in the list are not directly related.

Positional List example: database of related items - e.g. list of daily activities. Activities ‘next to each other’ in the list occur ‘next to each other’ in real life. New activity B is typically added/referred to as ‘activity that happens after A’.

Positional List ADT

Positional List ADT – abstraction of a linked list with "shielded" internal structure

Positional List ADT Implementation – linked list (SLL or DLL) is the natural choice for implementation of Node List ADT

- however, direct use of "node-based" operations should be avoided, because:

  (1) we should not expose too much information about the implementation of the list

  (2) we should not burden the end user with too much implementation details – e.g what to do with "links" on every insert / removal

SOLUTION: Use Position ADT – a helper ADT!

Positional List ADT (cont.)

```
public interface PositionalList<E> {
```

Generic Methods

```
public int size();
public boolean isEmpty();
```

Accessor Methods

```
public Position<E> first();
public Position<E> last();
public Position<E> before(Position<E> p) throws …;
public Position<E> after(Position<E> p) throws …;
```

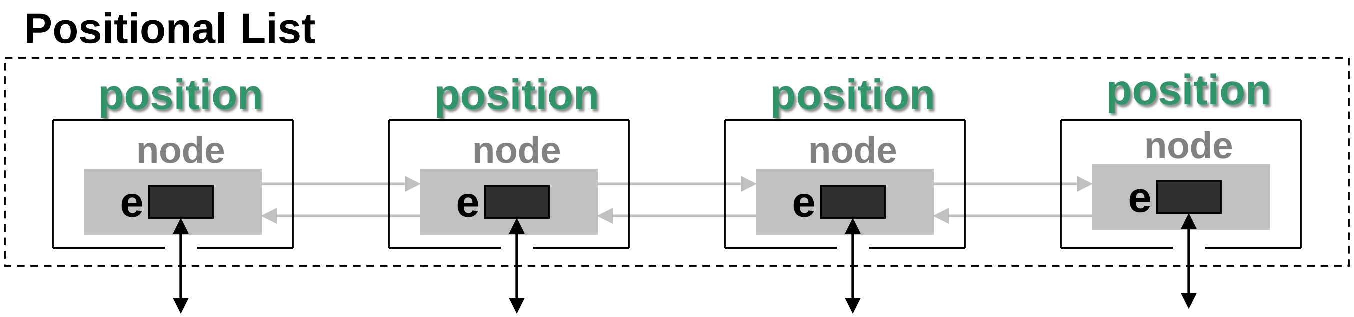
Update Methods

```
public Position<E> addBefore(Position<E> p, E e) …;
public Position<E> addAfter(Position<E> p, E e) …;
public Position<E> addFirst(E e);
public Position<E> addLast(E e);
public E remove(Position<E> p) throws …;
public E set(Position<E> p, E e) throws …;  }
```

Positional List ADT (cont.)

Position ADT – **encapsulates the idea of "node"** in a linked list

- **but, <u>has only one public method</u> *public E getElement()* which returns element stored at the given position**



None of get()/set() - next()/prev() methods of the underlying DNodes are accessible from the "outside" !!!

For outside user, Positonal List is viewed as a container of elements, which stores each element at/inside a position, and keeps positions arranged in a linear order.

NOTE: **Positions are defined relatively to their neighbours. The 'relative' neighbourhood of p does not change even if we replace or swap the element e stored at p with another element.**

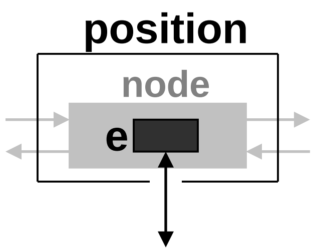
A position associated with element e does <u>NOT</u> change even if the rank of e changes in S. (Positions are NOT tied to rank!)

Positional List ADT (cont.)

```
interface Position<E> {
        E getElement() throws IllegalStateExcept..; }
```

Wrapper class interface that restricts access to the underlying DLLNode.

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

        private E element;
        private Node<E> prev;
        private Node<E> next;

        public Node(E e, Node<E> p, Node<E> n) {
             element = e;
             prev = p;
             next = n; }

        public E getElement() throws IllegalStateExcept... {
             if (next == null)              // convention for defunct node
               throw new IllegalStateException("Position no longer valid");
             return element; }
        ...
        // getPrev(), getNext(), setPrev(..), setNext(..), setElement(..)
}
```



Positional List ADT (cont.)

Example **[operations on a List]**

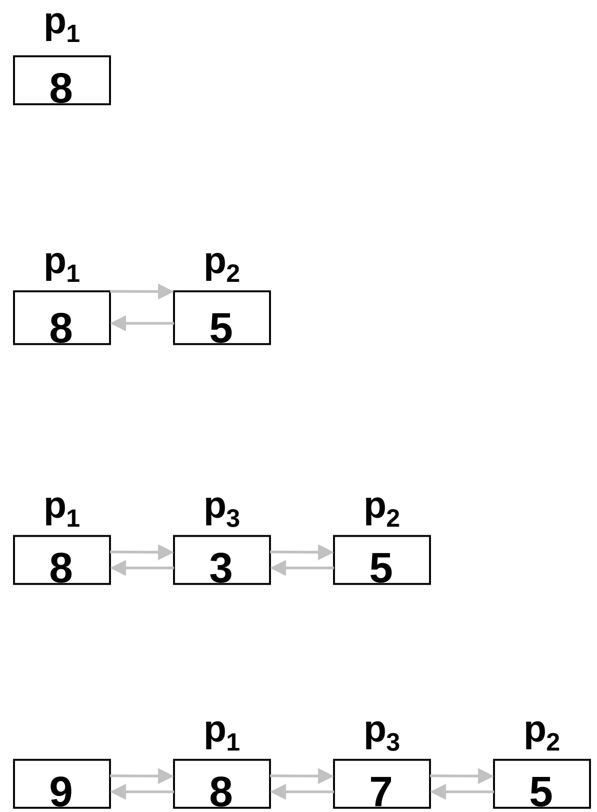
Method	Return Value	List Contents
addLast(8)	p	(8_p)
first( )	p	(8_p)
addAfter(p, 5)	q	$(8_p, 5_q)$
before(q)	p	$(8_p, 5_q)$
addBefore(q, 3)	r	$(8_p, 3_r, 5_q)$
r.getElement( )	3	$(8_p, 3_r, 5_q)$
after(p)	r	$(8_p, 3_r, 5_q)$
before(p)	null	$(8_p, 3_r, 5_q)$
addFirst(9)	s	$(9_s, 8_p, 3_r, 5_q)$
remove(last( ))	5	$(9_s, 8_p, 3_r)$
set(p, 7)	8	$(9_s, 7_p, 3_r)$
remove(q)	"error"	$(9_s, 7_p, 3_r)$

Positional List ADT (cont.)

Example **[operations on a List]**

p_1, p_2, p_3, … could as well be called a, b, c, d

Operation
addFirst(8)
p_1 = first()
addAfter(p_1, 5)
p_2 = next(p_1)
addBefore(p_2,3)
p_3 = prev(p_2)
addFirst(9)
set(p_3, 7)

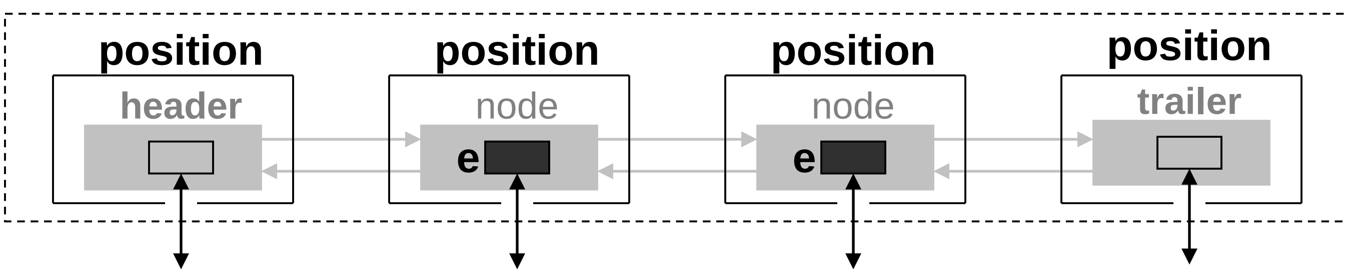


Positional List ADT: Doubly Linked List Implementation

DLL Implementation of Positional List ADT

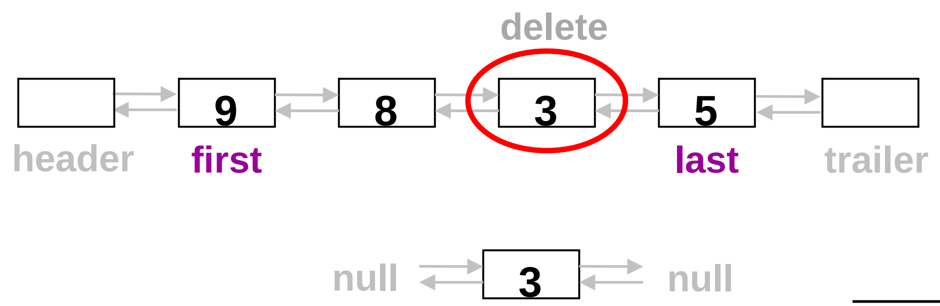
- DLL Nodes are viewed internally by List as ‘nodes’, from outside they are viewed only as positions
- through casting, given a position p, we can “unwrap” p to reveal the underlying node v

```
public class LinkedPositionalList<E> implements PositionalList<E> {
    private int size = 0;
    private Node<E> heaader, trailer;
    public LinkedPositionalList() {
        header = new Node<E>(null, null, null);
        trailer = new Node<E>(null, header, null);
        header.setNext(trailer);  }
```



Example Methods

```
/* checks if position is valid for this list, and if valid converts it to DNode */
private Node<E> validate(Position<E> p) throws IllegalArgumentExcep. {
        if ( !(p instanceof Node) ) throw new
                IllegalArgumentException("Invalid p.")

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

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

        return node;
}
```



Positional List ADT: Doubly Linked List Implementation (cont.)

```
package myList

public class NodePositionalList<E> {
    …
    private DNode<E> validate(Position<E> p) {
            …
    }
    …
}
```

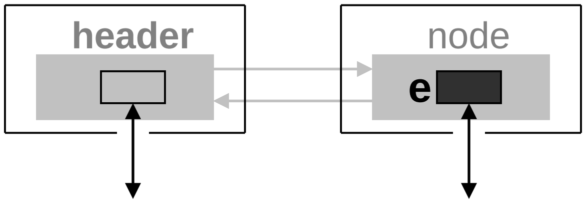
L.validate(p) cannot be run outside its own package

```
import myList.NodePositionalList<E>;

public class ListApplication<E> {
    NodePositionalList<E> L;
…
}
```

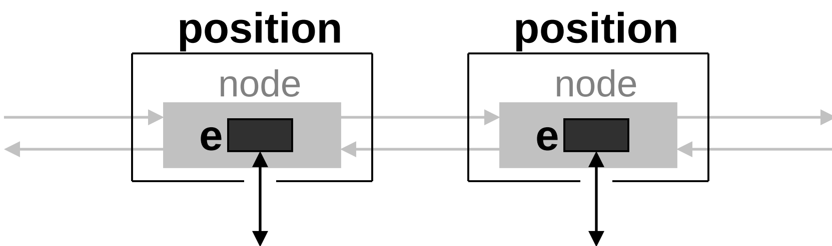
Positional List ADT: Doubly Linked List Implementation (cont.)

```
/* returns the given node as a Position or null, if it is a sentinel */
private Position<E> position(Node<E> node) {
    if (node == header || node == trailer)  return null;
    return node; }
```

```
/* returns the first Position in the list or null if empty */
public Position<E> first() {
    return position(header.getNext()); }
```

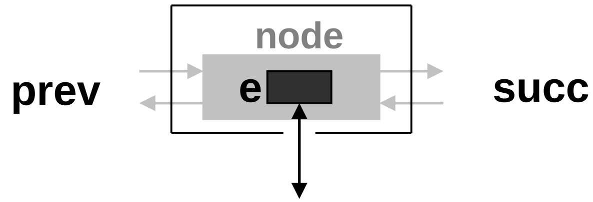



```
/* returns Position immediately before Position p or null if p is first */
public Position<E> before(Position<E> p) throws IllegalArgumentExcp. {
        Node<E> node = validate(p);
        return position(node.getPrev()); }
```



Positional List ADT: Doubly Linked List Implementation (cont.)

```
/* add element e to the linked list between the given nodes */
private Position<E> addBetween(E e, Node<E> prev, Node<E> succ) {
        Node<E> newest = new Node<>(e, prev, succ);
        prev.setNext(newest);
        succ.setPrev(newest);
        size++;
        return newest;
}
```



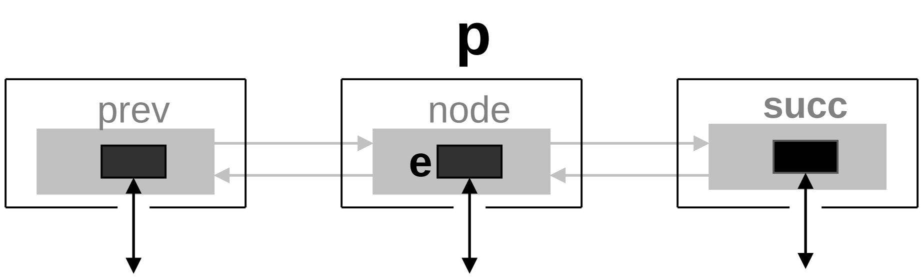

```
/* inserts element e at the front of the linked list and returns … */
public Position<E> addFirst(E e) {
        return addBetween(e, header, header.getNext());
}
```

Positional List ADT: Doubly Linked List Implementation (cont.)

```
/* inserts element e immediately before Position p and returns … */
public Position<E> addBefore(Position<E> p, E e)  throws IllegalArg.. {
        Node<E> node = validate(p);
        return addBetween(e, node.getPrev(), node);
}


/* replaces element stored at Position p and returns replaced element */
public E set(Position<E> p, E e)  throws IllegalArg.. {
        Node<E> node = validate(p);
        E answer = node.getElement(p);
        node.setElement(e);
        return answer;
}
```

Positional List ADT: Doubly Linked List Implementation (cont.)

```
/* removes element e stored at p, returns e, and invalidates p */
public E remove(Position<E> p)  throws IllegalArumentException {
        Node<E> node = validate(p);

        Node <E> pred = node.getPrev();
        Node <E> succ = node.getNext();

        pred.setNext(succ);
        suce.setPrev(pred);

        size --;

        E answer = node.getElement();

        node.setElement(null);
        node.setNext(null);
        node.setPrev(null);

        return answer;
}
```



Positional List ADT: Doubly Linked List Implementation (cont.)

Run Time – Good! all methods run in constant O(1) time

List Method	Time
size	O(1)
first	O(1)
last	O(1)
before	O(1)
after	O(1)
addBefore	O(1)
addAfter	O(1)
…	…

NOTE:

All methods run in O(1) time since they assume that the reference to the Position in question is given – no searching through DLL is required.