Position ADT (§ 5.2.2)
- The Position ADT models the notion of place within a data structure where a single object is stored.
- It gives a unified view of diverse ways of storing data, such as:
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object element(): returns the element stored at the position

List ADT (§ 5.2.3)
- The List ADT models a sequence of positions storing arbitrary objects.
- It establishes a before/after relation between positions.
- Generic methods:
  - size(), isEmpty()
- Accessor methods:
  - first(), last()
  - prev(p), next(p)
- Update methods:
  - replace(p, e)
  - insertBefore(p, e), insertAfter(p, e), insertFirst(e), insertLast(e)
  - remove(p)
- Generic methods:
  - size(), isEmpty()
Insertion

We visualize operation insertAfter(p, X), which returns position q

```
Algorithm insertAfter(p, e):
    Create a new node v
    v.setElement(e)
    v.setPrev(p)  // link v to its predecessor
    v.setNext(p.getNext())  // link v to its successor
    (p.getNext()).setPrev(v)  // link p's old successor to v
    p.setNext(v)  // link p to its new successor, v
    return v  // the position for the element e
```

Deletion

We visualize remove(p), where p = last()

```
Algorithm remove(p):
    t = p.element  // a temporary variable to hold the return value
    (p.getPrev()).setNext(p.getNext())  // linking out p
    (p.getNext()).setPrev(p.getPrev())
    p.setPrev(null)  // invalidating the position p
    p.setNext(null)
    return t
```
Performance

In the implementation of the List ADT by means of a doubly linked list:

- The space used by a list with $n$ elements is $O(n)$
- The space used by each position of the list is $O(1)$
- All the operations of the List ADT run in $O(1)$ time
- Operation `element()` of the Position ADT runs in $O(1)$ time