TEMA 3: LISTAS DE POSICIONES

Una **lista** es un **TAD** contenedor que consiste en una secuencia lineal de elementos. El acceso (búsqueda) a los elementos suele ser secuencial o lineal; y no está acotada (idealmente), es decir, puede crecer de acuerdo a las necesidades del programa y de la capacidad del computador. Algunos ejemplos son `IndexedList` y `PositionList` (lista de posiciones).

¿Qué inconvenientes tiene programar con `IndexedList<E>`? Se usan índices numéricos para acceder a los elementos y el hecho de insertar/borrar elementos es lento (*O(n)*)

Frente a ello, surge una lista más abstracta (`PositionList<E>`), ya que:

(1) No usa enteros como índices
(2) Usa objetos (nodos o posiciones) para trabajar con la lista, lo que proporciona una mayor persistencia
(3) No hay ninguna relación de orden entre dos objetos Position<E>
(4) El acceso y recorrido de los elementos de una PositionList<E> se hace usando objetos de tipo Position<E>

La interfaz `Position<E>` es la que usamos para representar el concepto de **nodo abstracto**. Únicamente permite acceder al contenido del **nodo**. No permite modificar su contenido ni cambiar su posición en la estructura de datos. Es la que utilizaremos más adelante en *árboles* y *grafos*.

```
public interface Position<E> {
     public E element();
}
```

La interfaz `PositionList<E>`, que hace uso de la interfaz `Position<E>`, es la siguiente:

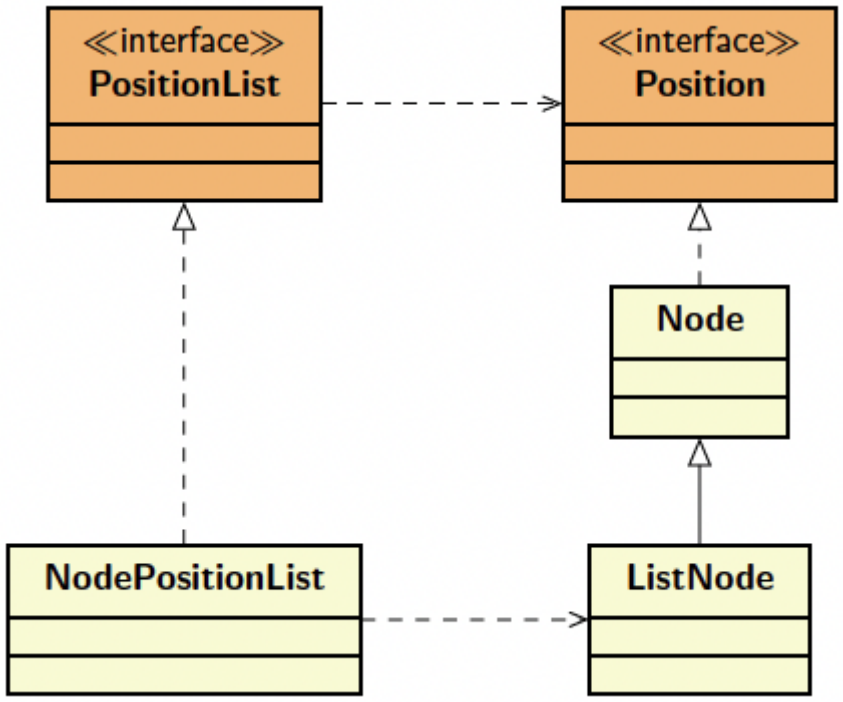
```
public interface PositionList <E> extends Iterable <E> {
     public int size ();
     public boolean isEmpty ();
     public Position <E> first ();
     public Position <E> last ();
     public Position <E> next ( Position <E> p)
           throws IllegalArgumentException ;
     public Position <E> prev ( Position <E> p)
           throws IllegalArgumentException ;
     public void addFirst (E elem );
     public void addLast (E elem );
     public void addBefore ( Position <E> p, E elem )
           throws IllegalArgumentException ;
     public void addAfter ( Position <E> p, E elem )
           throws IllegalArgumentException ;
     public E remove ( Position <E> p)
           throws IllegalArgumentException ;
     public E set( Position <E> p, E elem )
           throws IllegalArgumentException ;
     public Object [] toArray ();
     public E [] toArray (E[] a);
}
```

- El **recorrido** se hace usando bucles y nodos cursor de tipo `Position<E>`
- La **inicialización** consiste en hacer que el cursor apunte al primer nodo de la lista usando `l.first()` (o al último haciendo `l.last()`)
- Para **avanzar** moveremos el cursor a la siguiente posición con `l.next(cursor)` o a la anterior `l.prev(cursor)`
- La **condición de parada** depende del problema, pero suele incluir la condición de rango `cursor != null`
  NOTA: OJO con los posibles elementos **null**, ya que pueden hacer saltar la excepción *NullPointerException*

Ejemplo: mostrar los elementos de una lista con un while:

```
public static <E> void show ( PositionList <E> list ) {
  Position <E> cursor = list . first ();          // i = 0
  while ( cursor != null ) {                      // i<l. size ()
    System.out.println ( cursor . element ());    // print (l.get(i))
    cursor = list . next ( cursor );                    // i++
  }
}
```

Implementación de `Position<E>` y `PositionList<E>`



`Node<E,O> implements Position<E>`

Tiene dos atributos: `final O owner` (en qué lista está el nodo. Al ser *final* no se puede reutilizar en otra lista), `E elem` (contenido del nodo)

Tiene tres métodos: `n1.kindOf(n2)` (indica si `n1` y `n2` son parientes - tienen el mismo `owner` -), `checkNode(Position<E> p)` (comprueba si la posición `p` es pariente de `this` - el nodo -), `setElement(E element)` (asigna a `elem` el valor `element`)

`ListNode<E> extends Node<E,PositionList<E>>`

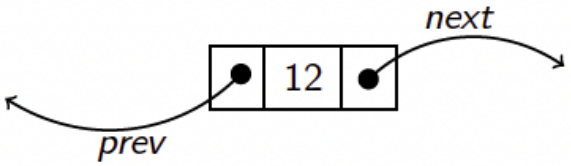
Usa la implementación de `NodePositionList<E>`

Tiene dos atributos más: `prev`, `next`

Tiene *getter* y *setter* para los atributos `prev`, `next`, `elem`

`setPrev` y `setNext` pueden recibir **null**

Ejemplo de una lista con un elemento 12



`NodePositionList<E> implements PositionList<E>`

Es una lista de nodos doblemente enlazada

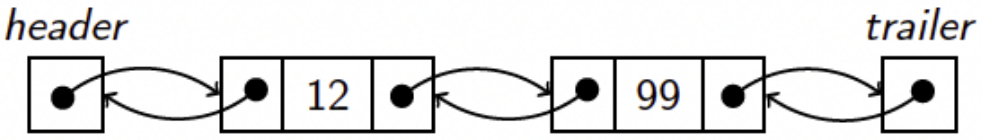
Tiene 3 atributos: el tamaño, y 2 nodos especiales (`header` y `trailer`)

Tiene 3 constructores: vacío, un array y una lista

Presenta limitaciones de tamaño por el tipo del atributo `size` y la memoria disponible

El método privado `checkNode` comprueba si una posición `p` es realmente un nodo válido, es decir, no es **null**, es de la clase `ListNode<E>` y es un nodo de la lista (usando `header.checkNode(p)`)

Ejemplo de una lista con dos elementos 12 y 99

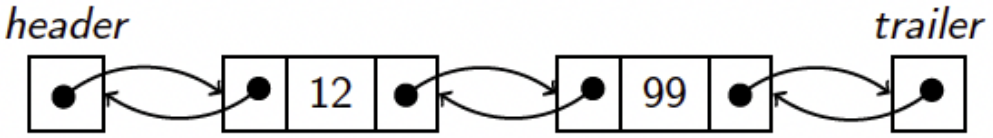


Ejemplo: `addFirst` en código Java, asumiendo que `size()>0`

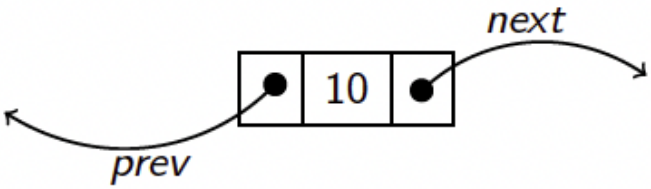
```
addFirst (ListNode <E> v) {
  ListNode<E> first = header.getNext();
  v.setNext(first);
  v.setPrev(header);
  first.setPrev(v);
  header.setNext(v);
  size++;
}
```

Ejemplo: `l.addFirst(10)` de forma visual

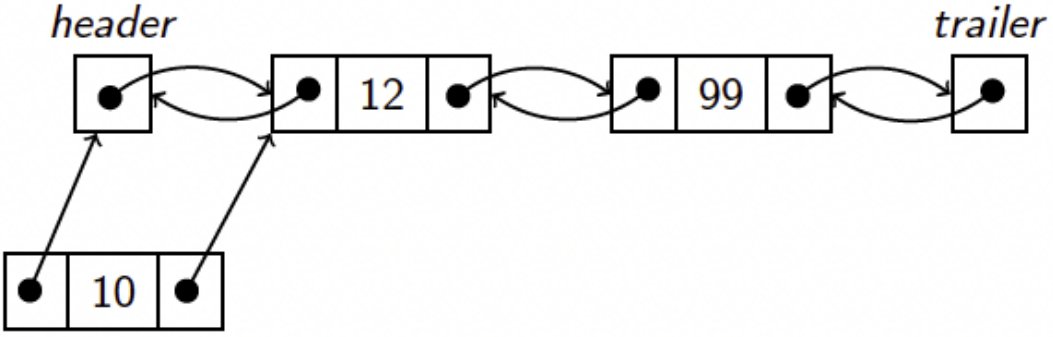
(1) Lista `l` inicial



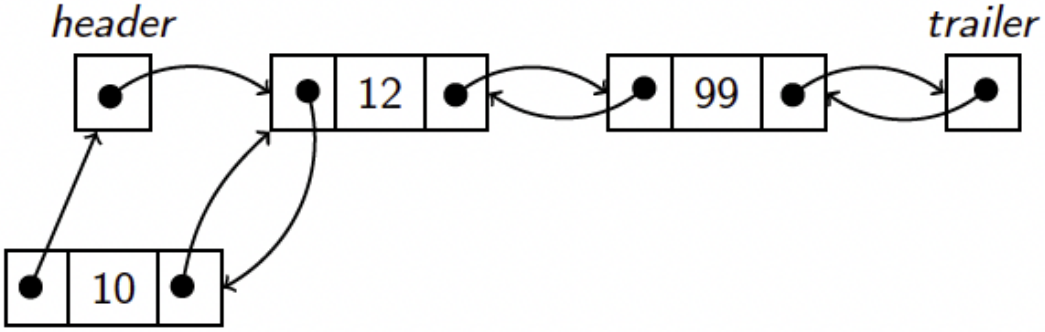
(2) Creamos un nuevo nodo `n` con el elemento 10



(3) Asignamos los atributos `prev` y `next` de `n`



(4) Asignamos el `prev` de `header.getNext()` a `n`



(5) Dejamos que `header` apunte a `n`