UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Tema 5. Tipos de datos lineales

Programación (PRG)

Jorge González Mollá

Departamento de Sistemas Informáticos y Computación

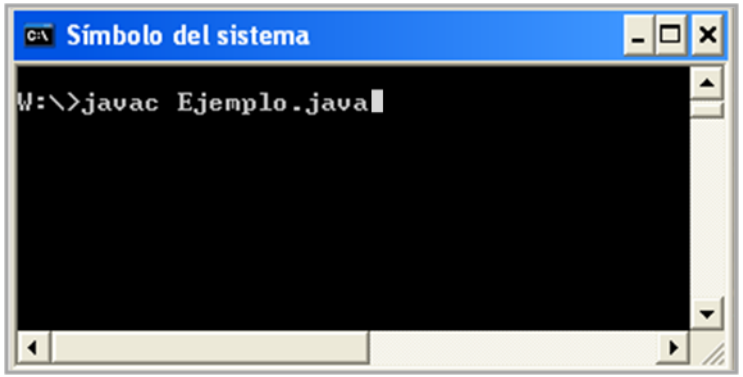
Índice

1. Introducción
2. Secuencias
   1) Recorrido y Búsqueda
   2) Inserción y Borrado
3. Estructuras de Datos Lineales
   1) Pilas
   2) Colas
   3) Listas con Punto de Interés

Definición

- Una *lista* es una secuencia de datos en la que el acceso puede realizarse desde cualquier posición de la misma.
- Una *lista con punto de interés* es una lista en la que hay una posición de acceso directo, llamada punto de interés o cursor.
- Si la lista tiene n datos, n≥0, numerados de 0 a n-1, el cursor puede estar ubicado en:
  - una posición 0≤i≤n-1: *sobre el elemento i-ésimo*
  - la posición i=n: *a la derecha del todo*.

- Ejemplo: Lista de caracteres en la línea de comandos del sistema. El propio cursor de edición es el punto de interés.



Definición

- En una *lista con punto de interés* todas las operaciones de inserción, borrado, etc., se hacen respecto al punto de interés:

```
W:\_
W:\E_
W:\Eje_
W:\Ejen_
W:\Ejen
W:\Eje_
W:\Ejem_
```

- En el modelo de lista con punto de interés que se va a presentar, el cursor sólo se puede mover hacia la derecha, o llevarlo al inicio.

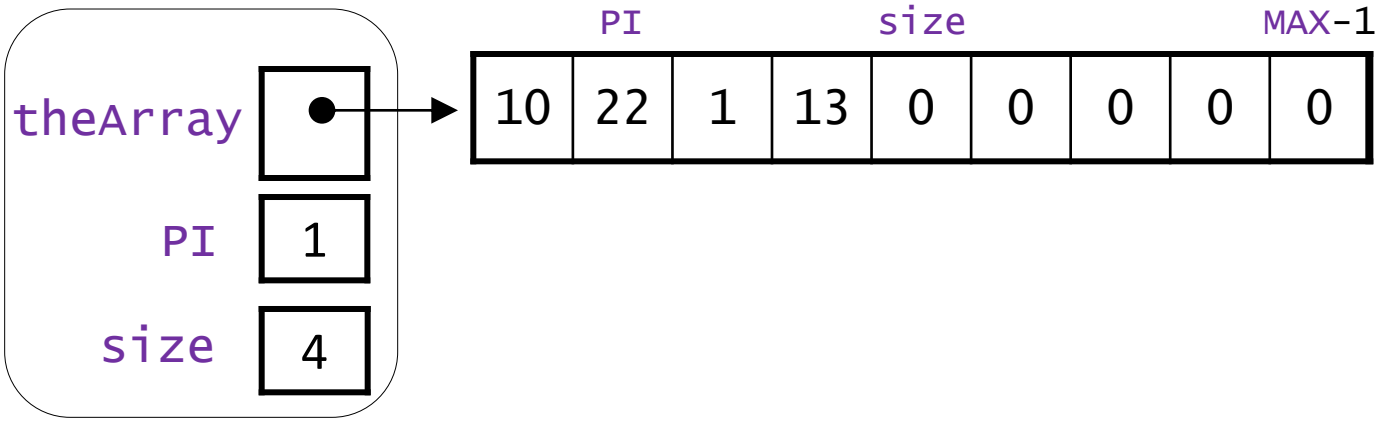
Interfaz de operaciones (API)

Operación	Descripción
public **ListPI**()	Crea una nueva **ListPI** vacía
public boolean **isEmpty**()	Devuelve `true` si la **ListPI** está vacía y `false` en caso contrario
public boolean **isEnd**()	Devuelve `true` si el cursor está al final de la **ListPI** y `false` en caso contrario
public int **size**()	Devuelve el nº de datos de la **ListPI** (≥0)
public void **begin**()	Sitúa el cursor al principio de la **ListPI**
public void **next**()	Desplaza el cursor una posición a la derecha de su posición actual *
public Tipo **get**()	Devuelve (sin eliminarlo) el elemento que está en el cursor *
public void **insert**(Tipo x)	Inserta x delante del cursor de la **ListPI**
public Tipo **remove**()	Elimina el elemento del cursor, devolviéndolo *

* Lanza `NoSuchElementException` si el cursor está al final de la **ListaPI**.

Implementación mediante arrays

- Los elementos de la lista ocupan las primeras posiciones consecutivas en el array. Se presenta una lista cuyos datos son `ints`:




```
public class ListPIIntArray {
   private int[] theArray;
   private int PI, size;
   private static final int MAX = ...;

   // Implementación de las operaciones:
   ...
}
```

Implementación mediante arrays

- Constructor `ListPIIntArray`: Crea la lista vacía.

```
public ListPIIntArray() {
    theArray = new int[MAX];
    size = PI = 0;
}
```

- Operación consultora `isEmpty`:

```
public boolean isEmpty() { return size == 0; }
```

- Operación consultora `isEnd`:

```
public boolean isEnd() { return PI == size; }
```

- Operación consultora `size`:

```
public int size() { return size; }
```

Implementación mediante arrays

- Operación `begin`: Sitúa el PI en la primera posición de la lista.

```
public void begin() {
   PI = 0;
}
```

- Operación `next`: Desplaza el PI una posición a la derecha.

```
public void next() {
   if (PI == size) { throw new NoSuchElementException(); }
   PI++;
}
```

- Operación `get`: Devuelve el elemento en el PI.

```
public int get() {
   if (PI == size) { throw new NoSuchElementException(); }
   return theArray[PI];
}
```

Implementación mediante arrays

- Operación `insert`: Inserta el elemento x en la posición del cursor. Hay que desplazar a la derecha todos los datos desde el cursor hasta el final de la lista.

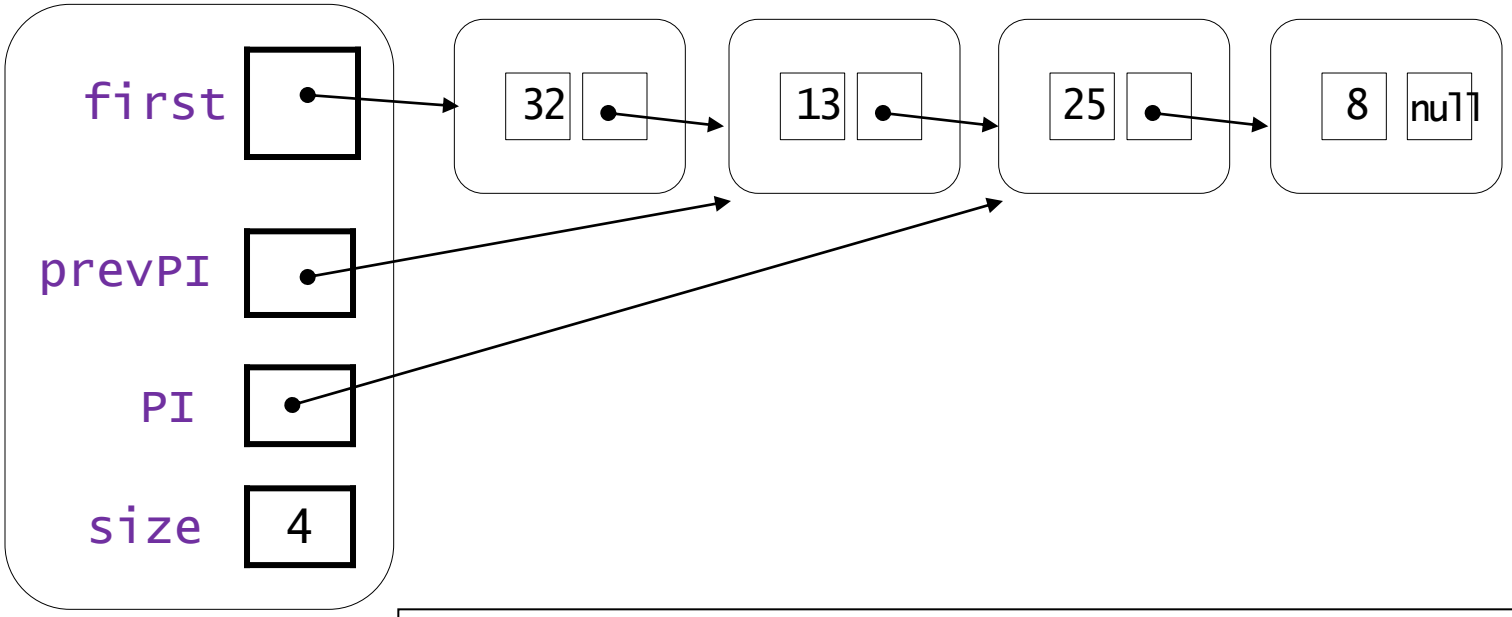
```
public void insert(int x) {
   if (size == theArray.length) {
      duplicateArray();
   }
   else {
      for (int k = size - 1; k >= PI; k--) {
         theArray[k + 1] = theArray[k];
      }
      theArray[PI] = x;
      PI++;
      size++;
   }
}
```

Implementación mediante arrays

- Operación remove: Devuelve y elimina de la lista el dato del cursor. Hay que desplazar a la izquierda todos los datos desde el cursor hasta el final de la lista.

```
public int remove() {
   if (PI == size) { throw new NoSuchElementException(); }
   int x = theArray[PI];
   for (int k = PI + 1; k < size; k++)
      theArray[k - 1] = theArray[k];
   size--;
   return x;
}
```

Implementación enlazada




```
public class ListPIIntLinked {
   private NodeInt first, PI, prevPI;
   private int size;

   // Implementación de las operaciones:
   ...
}
```

Implementación enlazada

- Constructor `ListPIIntLinked`: Crea la lista vacía.

```
public ListPIIntLinked() {
    first  = null;
    PI     = null;
    prevPI = null;
    size   = 0;
}
```

- Operación consultora `isEmpty`:

```
public boolean isEmpty() { return size == 0; }
```

- Operación consultora `isEnd`:

```
public boolean isEnd() { return PI == null; }
```

- Operación consultora `size`:

```
public int size() { return size; }
```

Implementación enlazada

- Operación `begin`: Sitúa el PI en la primera posición de la lista.

```
public void begin() {
   PI     = first;
   prevPI = null;
}
```

- Operación `next`: Desplaza el PI al siguiente nodo.

```
public void next() {
   if (PI == null) { throw new NoSuchElementException(); }
   prevPI = PI;
   PI     = PI.next;
}
```

- Operación `get`: Devuelve el elemento en el PI.

```
public int get() {
   if (PI == null) { throw new NoSuchElementException(); }
   return PI.data;
}
```

Implementación enlazada

- Operación `insert`: Crea un nuevo nodo para x delante del cursor. Si el cursor está al inicio, el nuevo nodo será el primero de la lista. En cualquier otro caso, el nuevo nodo se inserta entre el nodo cursor y su nodo predecesor.

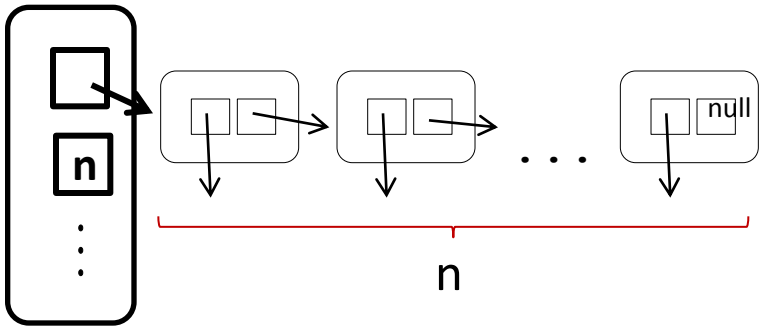
```
public void insert(int x) {
   NodeInt nuevo = new NodeInt(x, PI);
   if (PI == first) {
      first  = nuevo;
   } else {
      prevPI.next = nuevo;
   }
   prevPI = nuevo;
   size++;
}
```

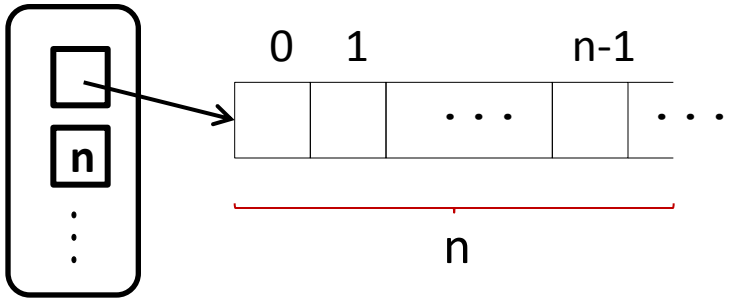
Implementación enlazada

- Operación `remove`: Devuelve y borra de la lista el elemento del PI. Si el cursor está al inicio, primero se actualiza a su nodo siguiente. En cualquier otro caso, los nodos anterior y posterior al nodo cursor quedan enlazados.

```
public int remove() {
   if (PI == null) { throw new NoSuchElementException(); }
   int x = PI.data;
   if (PI == first) {
      first = first.next;
   }
   else {
      prevPI.next = PI.next;
   }
   PI = PI.next;
   size--;
   return x;
}
```

Comparación de implementaciones





Operación	Representación enlazada	Representación con arrays
remove()	$\Theta(1)$	$O(n)$, $\Omega(1)$
insert(int x)	$\Theta(1)$	$O(n)$,$\Omega(1)$

- El resto de operaciones definidas para listas con punto de interés tienen el mismo coste constante, $\Theta(1)$, en las dos representaciones.

Implementación enlazada. Ampliación

- Añadir los siguientes métodos a la API de lista con punto de interés:

  a) Método que devuelva si `x` está o no en la lista (`boolean`). Si `x` se encuentra, sitúa el `PI` a la primera ocurrencia de `x`. Si no aparece, el `PI` no se mueve.
  b) Método que devuelva si `x` está o no en la lista (`boolean`) desde el cursor en adelante. Si `x` aparece, avanza el `PI` a `x`. Si no aparece, el `PI` no se mueve.

  Realizar su implementación en la clase `ListPIIntLinked`.

Implementación enlazada. Ampliación

- El siguiente método auxiliar busca la primera ocurrencia de x desde el nodo siguiente a ant en adelante; si tiene éxito, mueve el PI al nodo que la contiene:

```
private boolean buscar(NodeInt ant, int x) {
        NodeInt aux;
        if (ant == null) { aux = first; }
        else             { aux = ant.next; }
        while (aux != null && aux.data != x) {
             ant = aux; aux = aux.next;
        }
        if (aux == null) { return false; }
        else {
            prevPI = ant; PI = aux; return true;
        }
   }
```

- Los métodos pedidos son:

```
public boolean buscarInicio(int x) {
  return buscar(null, x); // null es el anterior a primero
}

public boolean buscarSiguiente(int x) {
  return buscar(prevPI, x); // prevPI es el anterior a PI
}
```