Tema 4: Representación Enlazada

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Tema 4- Representación Enlazada

Índice general:

Objetivos y Bibliografía

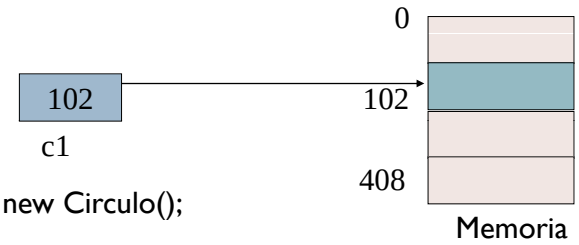
- Representar en Java un grupo de objetos mediante una lista enlazada de nodos.
- Confrontar las representaciones enlazada y secuencial.
- Insistir en las consecuencias del uso del polimorfismo para la representación enlazada de objetos.
- Introducir e implementar diversos tipos de representaciones enlazadas en base a criterios de eficiencia y claridad en su diseño.
- Bibliografía:
  - Weiss, M.A. Estructuras de datos en Java. Adisson-Wesley, 2000. Capítulo 16, Apartados 16.1 (sección 1.1) y 16.3

Variables Referencia y Objetos Java

1. Una variable Java (de un tipo no primitivo) referencia a un objeto de dicho tipo, pero no es un objeto de dicho tipo.
   - Por ejemplo, en la instrucción Circulo c1 = new Circulo();
     - c1 referencia al Objeto creado por *new Circulo()*
     - c1 es una variable referencia de tipo (estático) Circulo
2. Una referencia contiene la dirección de memoria donde se almacena el objeto al que referencia (tras haberlo instanciado vía new).
   - Por ejemplo, null es una dirección de memoria predefinida, que sirve para representar que, si c1 = null
     - c1 no referencia a ningún objeto
     - c1 es una variable referencia nula

Referencias en Java

- Una variable Referencia almacena la dirección de memoria en la que se encuentra el Objeto al que referencia.



Circulo c1 = new Circulo();

- Operaciones sobre una variable referencia:
  - las que permiten examinar o manipular su valor (=, ==, !=)
  - las que permiten examinar o manipular el estado del Objeto al que referencia (Casting, . , instanceof).

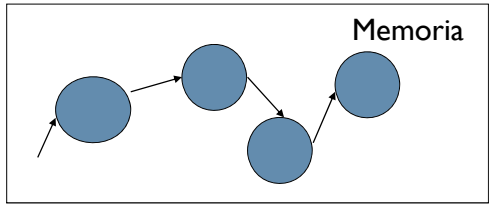
Representación Secuencial

- La gestión más sencilla de un grupo de objetos se puede conseguir usando una representación secuencial.
  - Basada en un vector (Object [])

- Características principales:
  - Ventajas:
    - Acceso en tiempo constante si conocemos la posición del objeto.
    - No introduce sobrecarga adicional en el almacenamiento de datos.
  - Inconvenientes:
    - Requiere conocer a priori el número máximo de elementos (reserva de memoria inicial).
    - La inserción/borrado de manera ordenada puede requerir el desplazamiento de otras componentes para mantener la representación secuencial.

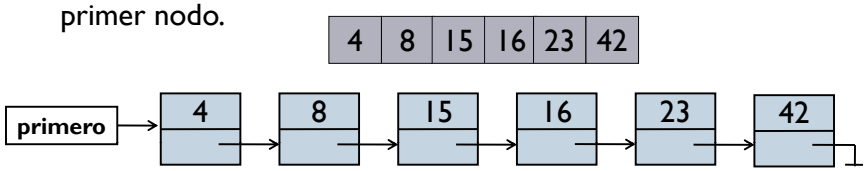
Representación Enlazada: Motivación

- Ofrece una alternativa a la representación secuencial para la gestión de un grupo de Objetos.
- Objetivos principales:
  1. **Reducir el coste** de las operaciones de **inserción y borrado** de un Objeto del grupo.
  2. Eliminar la necesidad de reserva inicial de memoria.

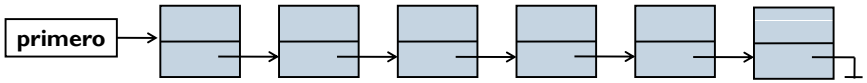


Representación Enlazada de un Grupo de Elementos

- La Lista Enlazada consta de una secuencia enlazada de nodos situados en la memoria dinámica.
- Cada nodo representa a un elemento del grupo y por tanto se compone de:
  - Un dato del tipo base del grupo.
  - Una referencia al siguiente nodo de la lista
- El campo siguiente del último nodo es una referencia a null;
- El acceso a la Lista Enlazada se realiza vía la referencia al primer nodo.

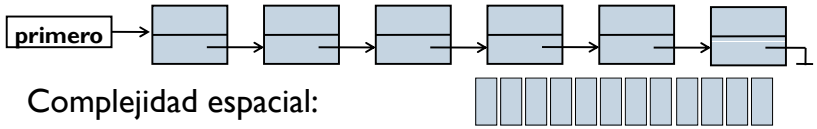


Complejidad de una Lista Enlazada de Nodos



- Complejidad Espacial (Consumo de memoria):
  - Proporcional al número de nodos de la Lista Enlazada, es decir, **Lineal con la Talla del Grupo**.
- Complejidad Temporal:
  - Búsqueda de un nodo: Coste lineal con la talla de la lista.
  - Inserción o borrado de un nodo: **Independiente de la Talla del Grupo (constante)**, sin necesidad de desplazar nodos, sólo modificando los enlaces adecuados.
    - Aunque buscarlo sigue precisando un coste lineal con el número de nodos.

Representación Enlazada VS Secuencial



- Complejidad espacial:
  - Un array no admite redimensionamiento dinámico (requiere conocer el máximo número de elementos).
  - En la enlazada no hay reserva a-priori de memoria y puede crecer dinámicamente. Hace falta memoria para representar los enlaces.
- En complejidad temporal,
  - Acceso a un elemento cuya posición es conocida: por contigüidad, mejor la secuencial.
  - Inserción o borrado de un elemento: por ausencia de desplazamientos (mantener la contigüidad), mejor la Enlazada.

Listas Enlazadas Genéricas

- En una Lista Enlazada Genérica (LEG), sus nodos pertenecen a una clase (**NodoLEG<E>**) que tiene dos atributos:
  1. **E dato**, que representa la información asociada a cualquier objeto de tipo genérico E.
  2. **NodoLEG<E> siguiente**, que representa una referencia al siguiente nodo de la Lista Enlazada.
- La utilización de la genericidad permite:
  - Definir nodos que almacenarán un tipo de datos.
  - Construir una lista homogénea (mismo tipo) de nodos.
  - Detectar incoherencias de tipos en tiempo de compilación.
    - i.e.: Instanciar un Nodo de String y tratar de albergar un Integer

La Clase de los Nodos de una LEG: NodoLEG

```
package librerias.estructurasDeDatos.lineales;
class NodoLEG<E>{
  E dato;
  NodoLEG<E> siguiente;

 NodoLEG(E dato) {
    this(dato, null);
  }
 NodoLEG(E dato, NodoLEG<E> siguiente) {
    this.dato = dato;
    this.siguiente = siguiente;
  }
}
```

La Clase que Representa una LEG

```
package librerias.estructurasDeDatos.lineales;
import excepciones.*;
public class LEG<E>{
  protected NodoLEG<E> primero;  protected int talla;
  public LEG() { ... }
  public int talla() { ...}
  public void insertar(E x) {...}
  public void insertarEnFin(E x) { …}
  public void insertar(E x, int i){ …}
  public boolean eliminar(E x) { ...}
  public E recuperar(int i);
  public int indiceDe(E e){ … }
  public E[] toArray(E[] a){ … }
  public String toString() { ...} }
```


Inicialización de una LEG: Constructor

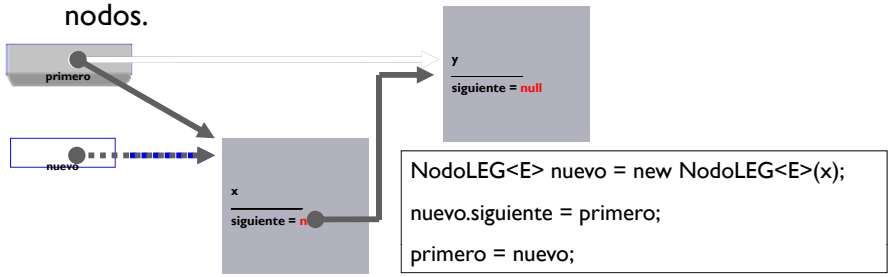
- Un constructor siempre sirve para dar valor a los atributos del objeto.
  - El atributo primero es una referencia al primer nodo de la lista y es el único punto de entrada para recorrer una lista.
  - Inicialmente, la lista está vacía y no hay ningún nodo inicial. Por ello, primero apunta a null

```
public LEG() {
    primero = null;
     this.talla = 0;
}
```

- Mecanismo de creación de una lista enlazada:
  - LEG<Integer> l1 = new LEG<Integer>();
  - LEG<Vehiculo> l2 = new LEG<Vehiculo>();

Inserción en una LEG: Estrategia

1. Como criterio, siempre se insertan los nodos en cabeza de la LEG. Al insertar siempre en la cabeza de la LEG:
   - El orden de los nodos en la lista es justo el inverso al de inserción.
   - El coste de inserción es independiente de la talla de la LE.
2. El acceso a la lista siempre se realiza desde el primer nodo. Es posible recorrerla a través de las referencias a los siguientes nodos.



Inserción en una LEG: Código

- Trasladamos a código la estrategia de inserción:

```
public void insertar(E x){
   NodoLEG<E> nuevo = new NodoLEG<E>(x);
   nuevo.siguiente = primero;
   primero = nuevo; this.talla++;
}
```

- Es posible compactar el código:

```
public void insertar(E x){
   primero = new NodoLEG<E>(x, primero); this.talla++;
}
```

- Uso de la genericidad:
  - Dado que el código pertenece a la clase LEG<E>, es posible referenciar a E

Inserción al Final

```
public void insertarEnFin (E  x ){
     NodoLEG<E> nl = new NodoLEG<E>(x); this.talla++;
     NodoLEG<E>  aux  = primero;
     if (aux == null) primero = nl;
     else {
        while ( aux.siguiente != null )  aux = aux.siguiente;
       // aux referencia al último nodo de la lista
        aux.siguiente = nl;
           }
}
```

- Casos especiales:
  - Relacionados con las modificaciones a los principales atributos de la lista (en este caso, con la referencia primero).

Mostrando los Datos de la LEG: toString

```
public String toString() {
 String res = "";
 for ( NodoLEG<E> aux = primero; aux != null; aux = aux.siguiente )
       res += aux.dato.toString()+"\n";
  return res;
}
```

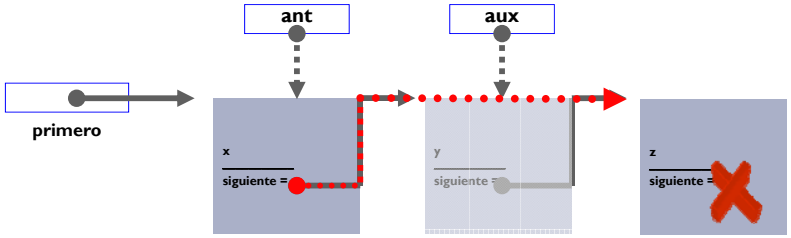
- Nótese la analogía entre recorrer una Lista Enlazada y recorrer un array:

```
for (int aux = 0 ;  aux != v.length;  aux++)
```

¿Cómo se haría un recorrido descendente?

Borrado de un Nodo en una Lista Enlazada (I)

- El borrado es un caso particular de búsqueda de un elemento.
- Caso General:
  - Borrado de un nodo intermedio:



- Caso particular: **ant.siguiente = aux.siguiente**
  - Borrado del primer nodo (ant == null) : primer = aux.siguiente

Borrado de un Nodo en una Lista Enlazada (II)

```
public boolean  eliminar(E x) {
    NodoLEG<E>  aux = primero, ant = null; boolean res = false;
    while ( aux != null && !aux.dato.equals( x ) ) {
            ant = aux; aux = aux.siguiente;
     }
    if ( aux != null ) {
       res = true; this.talla--;
       if ( ant == null )  primero = aux.siguiente;
        else ant.siguiente = aux.siguiente;
    }
     return res;
}
```

Recuperación de un Elemento de la Lista

```
/** SII talla()>0 AND 0<=i<talla **/
public E recuperar(int indice) {
        NodoLEG<E> aux; int j;
        for ( aux = primero, j = 0; j < i ; aux = aux.siguiente, j++ ){;}
        return aux.dato;
}
```

- Obtiene el objeto del nodo que ocupa i-ésima posición.
- La precondición debe satisfacerse para que el método funcione correctamente.

El método toArray

```
public E[] toArray(E[] a){
    NodoLEG<E> aux; int i;
    for ( aux= primero, i = 0; aux != null; aux = aux.siguiente, i++)
        a[i] = aux.dato;
    return a;
}
```

- Se utilizaría de la siguiente manera:
  - LEG<Manzana> l = new LEG<Manzana>();
  - //Inicialización de l
  - Manzana[] v = new Manzana[l.talla()];
  - v = l.toArray(v);

Uso de una LEG

```
import lineales.*;
import excepciones.*;
public class TestListaInteger {
    public static void main(String args[]){
        LEG<Integer> l = new LEG<Integer>();
        l.insertar(new Integer(9));
        l.insertar(new Integer(12));
        System.out.println("Lista de Integer actual:\n"+l.toString());
        System.out.print("Borrando de la Lista el 10: ");
        if (!l.eliminar(new Integer(10)))
            System.out.println("Elemento inexistente.");
     }
}
```