Grado en Ingeniería Informática
Grado en Matemáticas e Informática

Asignatura: PROGRAMACIÓN II

Implementación de contenedores

Ángel Lucas González Martínez
Jaime Ramírez Rodríguez

DLSIIS - E.T.S. de Ingenieros Informáticos
Universidad Politécnica de Madrid

Noviembre 2013

Implementación de contenedores en Java

Cadenas enlazadas

Cadenas enlazadas

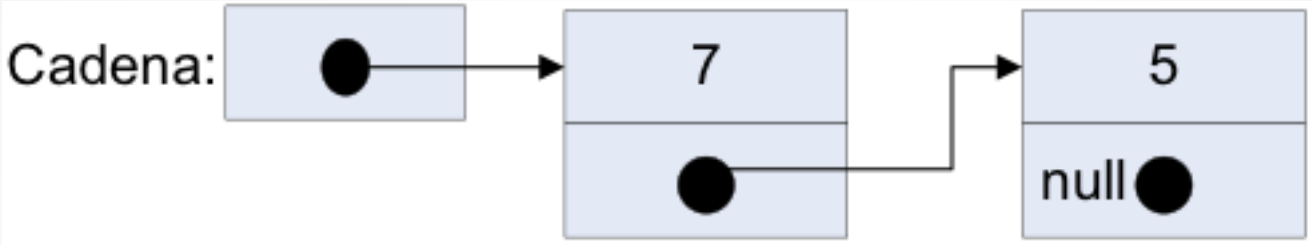
- Se utilizan para almacenar secuencias de datos cuyo tamaño máximo no se conoce al iniciarse el programa.
- Problema: ordenar una secuencia de enteros dada por el teclado.
  - ¿Dónde guardamos esta secuencia?

```
int secuencia[] = new int[TAMMAX]; // ¿TAMMAX?
```

  - No sabemos qué valor usar para TAMMAX: si usamos un valor muy grande, desperdiciaremos el espacio, y si usamos uno pequeño, nos podemos quedar cortos.
- Solución: estructura de datos de tamaño variable.
  - El único límite vendrá dado por la memoria del ordenador
  - La estructura tendrá el tamaño necesario en cada momento, ni más ni menos.

Cadenas enlazadas

- Ejemplo: cadena enlazada para la secuencia <7, 5>:



- La cadena está formada por nodos.
- Cada nodo contiene un dato, y una referencia al siguiente nodo (si no existe, la referencia vale **null**).
- Para acceder a los nodos de la cadena, necesitamos un puntero al primer nodo.
  - Cada vez que queramos realizar una operación sobre la cadena debemos proporcionar este puntero.
- La cadena vacía se representa como una referencia con valor **null**.

Implementación en Java

```
public class Nodo<Informacion> {
  private Informacion dato;
  private Nodo<Informacion> siguiente;
  public Nodo(Informacion dato) {
    this.dato=dato;
  }
  public Nodo(Informacion dato, Nodo<Informacion> siguiente) {
    this.dato=dato;
    this.siguiente=siguiente;
  }
  public Nodo<Informacion> darSiguiente() {
    return this.siguiente;
  }
  public Informacion darDato() {
    return this.dato;
  }
  public void fijarSiguiente (Nodo<Informacion> siguiente) {
    this.siguiente=siguiente;
  }
}
```

Implementación en Java

```
public class CadenaEnlazada<Informacion> {
  private Nodo<Informacion> cabeza;
  public CadenaEnlazada() { // constructor: crea cadena vacía
    setCabeza(null);
  }
  public void setCabeza(Nodo<Informacion> cabeza) {
    this.cabeza = cabeza;
  }
  public Nodo<Informacion> getCabeza() {
    return this.cabeza;
  }
}
```

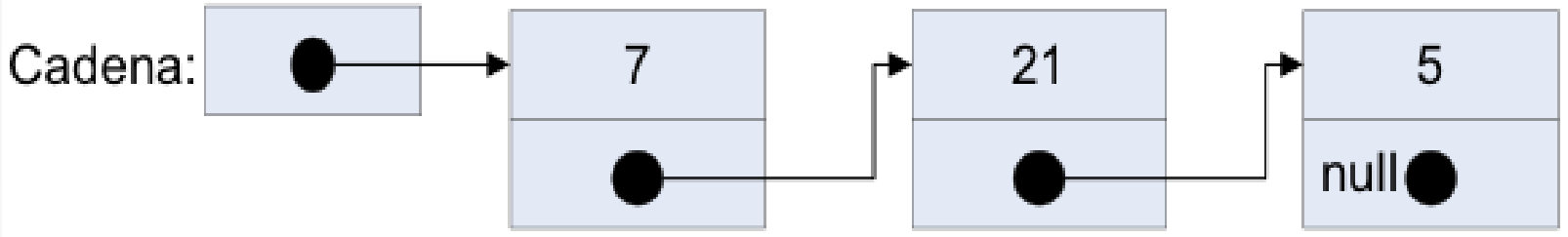
Ejemplo de manejo (I)

- El código para crear la cadena enlazada para la secuencia <7, 5> sería:

```
public class PruebaCadena {
  public static void main(String[] args) {
    CadenaEnlazada<Integer> cadena = new CadenaEnlazada<Integer>();

    cadena.setCabeza(new Nodo<Integer>(7, null));
    Nodo<Integer> cabeza = cadena.getCabeza();
    cabeza.fijarSiguiente(new Nodo<Integer>(5, null));
    // equivale a esto:
    // cadena.setCabeza(
         new Nodo<Integer>(7, new Nodo<Integer>(5, null))
       );
    }
  }
}
```

Ejemplo de manejo (II)

- Supongamos que queremos introducir un elemento en la cadena anterior <7, 5> y obtener <7, 21, 5>:



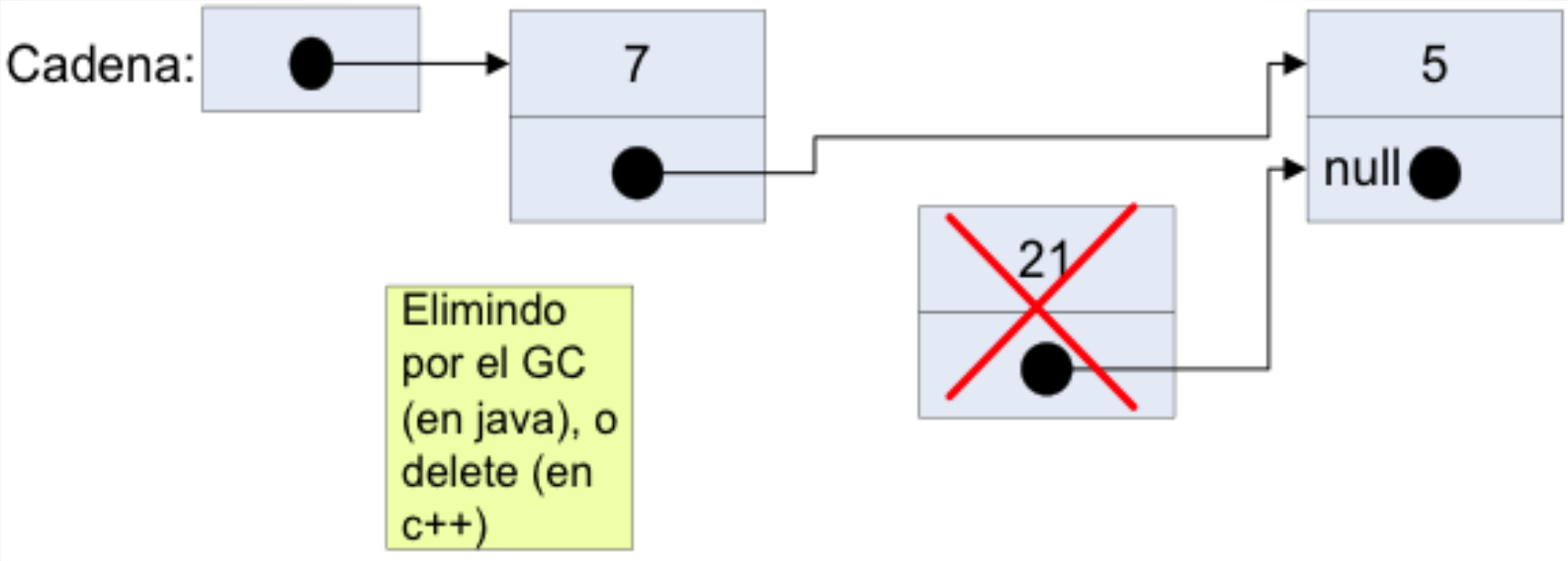
- El código podría ser el siguiente:

```
cabeza = cadena.getCabeza();
cabeza.fijarSiguiente(
   new Nodo<Integer>(21, cabeza.darSiguiente())
);
```

Ejemplo de manejo (III)

- Y para eliminarlo y obtener de nuevo < 7, 5 >:

```
cabeza = cadena.getCabeza();
cabeza.fijarSiguiente(cabeza.darSiguiente().darSiguiente());
```



Otras clases de cadenas enlazadas

- Las cadenas enlazadas se emplean para implementar secuencias.
- Las cadenas simplemente enlazadas presentan algunas carencias con respecto a la complejidad de algunas de sus operaciones (por ejemplo, longitud, inserción por el final y concatenación).
- Existen distintas modificaciones de las cadenas simplemente enlazadas para eliminar esas desventajas:
  - Mantener la **longitud como dato precomputado** (cadenas enlazadas con longitud).
  - Mantener un **puntero al último nodo** de la cadena (cadenas enlazadas con puntero al último y cadenas enlazadas circulares).
- Otra variante: cadenas doblemente enlazadas.

Implementación de contenedores clásicos

Implementaciones de Pilas y Colas

- Implementación del contenedor Pila
  - Clase genérica **Pila<Informacion>**: cadena enlazada
- Implementación del contenedor Cola
  - Clase genérica **ColaSimple<Informacion>**: cadena enlazada.
  - Clase genérica **ColaPrimeroUltimo<Informacion>**: cadena enlazada con puntero al último.
  - Clase genérica **ColaCircular<Informacion>**: cadena enlazada circular.
- Implementación del contenedor Pila Acotada
  - Clase genérica **PilaAcotada<Informacion>**: array + indiceCima

Pila Acotada

Pilas acotadas

- Si restringimos el número máximo de elementos de una pila obtenemos una pila acotada.
  - Cambia el constructor para recibir el tamaño
- El contenedor Pila Acotada es un contenedor distinto al contenedor Pila:
  - se introduce una nueva operación (*estaLlena*) y,
  - **más importante que introducir una operación nueva**, se **restringe** la precondición de una operación ya existente (no se puede apilar si la pila está llena).

Pilas acotadas en Java

- Se utiliza una clase genérica cuya variable de tipo (*Información*) representa el tipo de los elementos.
- Los **elementos** se guardan en un atributo *array* con la capacidad máxima especificada en el constructor.
- Se usa un atributo **indiceCima** que indica la siguiente posición libre en el *array*.
  - Se inicializa a cero.
  - Se incrementa cada vez que se apila un elemento.
  - Se decrementa cada vez que se desapila un elemento.

Implementación de contenedores en Java

Contenedor Lista

Contenedor Lista

- Ya hemos estudiado algunos contenedores (Pilas, Colas) que pueden ser utilizados para almacenar colecciones de datos.
- Ahora bien, un programa cliente no puede recorrerlas sin destruirlas.
  - Por ejemplo: para buscar un elemento, etc.
- El contenedor Lista va a admitir recorridos que no modifiquen la estructura.

Contenedor Lista

- Representa una secuencia de valores que admite habitualmente las siguientes operaciones:
  - *Constructor de lista vacía*, *estaVacia*, *primero*, *insertar* (al principio), *borrar* (el primero), *concatenar*.
- Nuestra implementación del contenedor Lista incluirá algunas operaciones más que facilitarán su utilización: *equals*, *clone*, *vaciar*, etc.

Implementación del contenedor Lista

- Clase genérica ListaIterable<Informacion> basada en una cadena enlazada.
- **Recorrido de una lista:** se va a utilizar un iterador interno (atributo actual).
  - Un **iterador** es una referencia a un elemento de la lista que podemos mover a lo largo de la lista para acceder a cada uno de los elementos.
  - **Operaciones necesarias:** alPrincipio(), siguiente(), haySiguiente(), getActual()

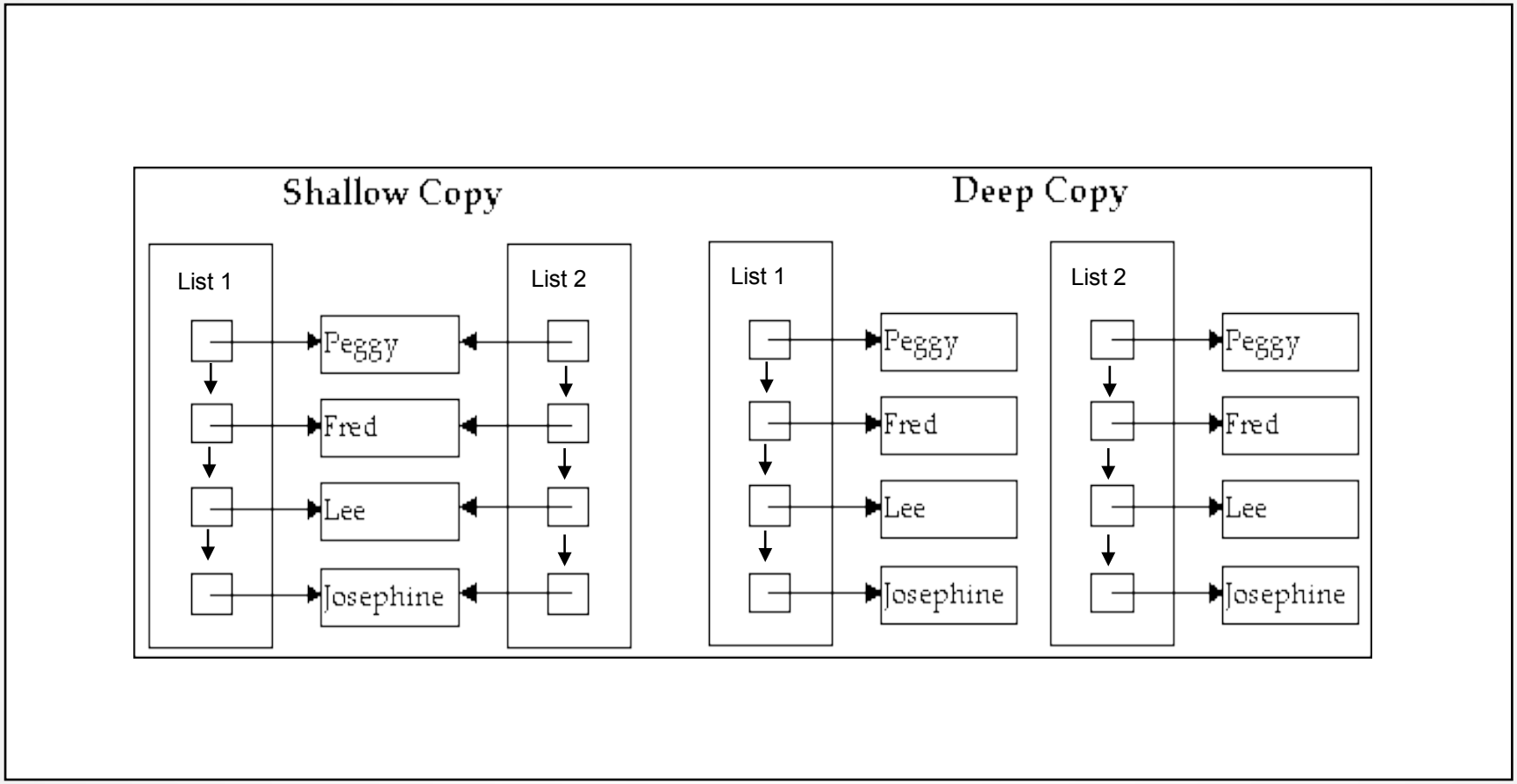
```
// PRE: lista no está vacía

lista.alPrincipio();
while (lista.haySiguiente()) {
   System.out.println(lista.getActual());
   lista.siguiente();
}
System.out.print(lista.getActual());
```

Implementación del contenedor Lista

- El constructor ListaIterable(ListaIterable lista) devuelve una ***shallow copy*** de la lista.
  - Una ***shallow copy*** es una copia que tiene sus propios nodos, pero comparte los objetos dato con la lista original.
  - Los datos no se pueden duplicar dentro del código de la lista porque el tipo genérico Información no garantiza que exista un constructor o método con un nombre predefinido que permita obtener la copia.
  - La decisión de que la copia de la lista sea una ***shallow copy*** es la misma que se ha adoptado en la API de Java (véanse las clases LinkedList y ArrayList).

Implementación del contenedor Lista

- *Shallow Copy* Vs *Deep Copy*

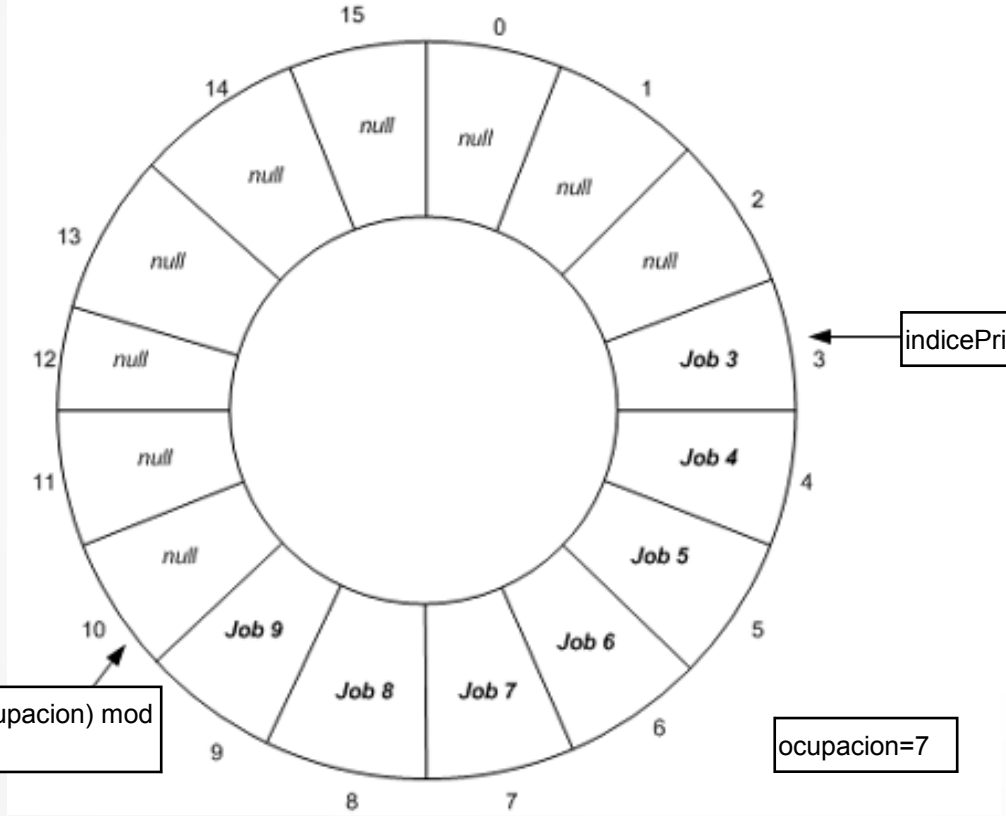


Contenedor Buffer (acotado)

- Se puede ver como una cola acotada.
- Los *buffers* se utilizan para que un proceso/dispositivo pueda mandarle datos a otro proceso/dispositivo sin necesidad de que la velocidad de escritura coincida con la de lectura en todo momento.
- Puede tener un tamaño limitado.
  - Sólo cuando se llena el proceso emisor debe esperar.
  - Sólo cuando se vacía el proceso receptor debe esperar.
- Típicamente incorpora las siguientes operaciones: *primero*, *insertar* (por el final), *borrar* (el primero), *darOcupacion*.

Implementación del contenedor Buffer

- Clase Buffer de Integers basada en un array circular.
- La clase genérica requiere reflexión en el constructor.



- Cada vez que se inserta un nuevo elemento:
  1. elementos[(indicePrimero + ocupacion) % Capacidad] = nuevoElemento
  2. ocupacion++
- Cada vez que se borra un elemento:
  1. ocupacion—
  2. indicePrimero=(indicePrimero +1)%Capacidad

Objetos compuestos:
Buenas prácticas al trabajar con atributos de tipo objeto

Inicializar atributos de tipo objeto

- Cada objeto debe manejar su propia copia de los datos (en objetos) a los que referencia desde sus atributos.
  - Excepto si estos datos son inmutables (clase String).
- Si el constructor (o un método set()) recibe como parámetro un objeto que debe asignar como valor inicial a un atributo:
  - El constructor debe crear y asignar una copia. Ejemplo:

```
public Persona(String nombre,
               FechaComparable fechaNacimiento) {
  this.nombre = nombre;
  this.fechaNacimiento =
               new FechaComparable(fechaNacimiento);
}
```

Acceso a atributos de tipo objeto

- Un objeto NO DEBE devolver una referencia a un objeto interno al que se referencia desde uno de sus atributos:

```
public FechaComparable getFechaNacimiento() {
  return this.fechaNacimiento;
}
```

- En su lugar debe devolver una referencia a una copia:

```
public FechaComparable getFechaNacimiento() {
    return  new FechaComparable(this.fechaNacimiento);
}
```

- De lo contrario, estaríamos permitiendo que otro objeto modifique sus datos privados.

¿Y si no se sabe duplicar el atributo?

- Tres opciones:
  - No se devuelve con get ni se pasa como parámetro en el constructor.
  - Si se pasa como parámetro en el constructor, nos aseguramos de que nadie más lo pueda referenciar:

```
new Persona( new Fecha(…), ...);
```

  - Si no queda más remedio, se devuelve la referencia al atributo, pero se documenta claramente (en el Javadoc, por ejemplo) para advertir sobre posibles efectos laterales al modificar los datos compartidos.