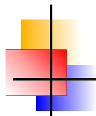


Algoritmos y Estructuras de Datos

Iteradores

Guillermo Román Díez

groman@fi.upm.es

Universidad Politécnica de Madrid

Curso 2015-2016

Iteradores

- Es muy común tener que iterar linealmente sobre todos los elementos de un TAD
  - Recordamos el código del método `show`

```
public static <E> void show(PositionList<E> list) {
  if (list == null) return;
  Position<E> cursor = list.first();
  while (cursor != null) {
    System.out.println(cursor.element());
    cursor = list.next(cursor);
  }
}
```

- Poblemas de este código
  - Es específico para la lista de posiciones
  - El programador "cliente" es el responsable de iterar adecuadamente

Iteradores

- Se puede abstraer el cursor para tener código reutilizable para otros TAD convirtiendo el cursor en un TAD llamado `Iterador`
- Un objeto `Iterador` permite iterar linealmente sobre los elementos de otro TAD
  - La iteración se realiza utilizando métodos del `Iterador`
  - No se usan métodos del TAD
  - Se puede reutilizar código para iterar otros TADs

```java
public static <E> void show(PositionList<E> list) {
  Iterator<E> it = list.iterator();    // Nos da un iterador
                                       // ya inicializado
  while (it.hasNext()) {               // Bucle mientras
                                       // hay mas elementos
     System.out.println(it.next());    // Cogemos el elemento
                                       // y queda el cursor
  }                                    // avanzado
}
```

Iterador

- El método `iterator` devuelve un iterador inicializado en el primer elemento de la estructura de datos
- `hasNext` devuelve `true` mientras haya algún elemento pendiente
- `next` devuelve el elemento accesible desde el cursor
  - Deja el cursor avanzado (post-incremento)
  - Esto se conoce como un “efecto secundario” o “side effect”
- Es análogo al recorrido de un array con post-incremento

```
int i = 0;
while (i < arr.length) {
  System.out.println(arr[i++]);
}
```

Ejemplo con otra estructura

Pregunta

¿cuáles serían los cambios sobre el método `show` para recorrer una estructura de datos de tipo `FIFO` o `LIFO`?

```
public static <E> void show(PositionList<E> list){
  Iterator<E> it = list.iterator();
  while (it.hasNext()) {
     System.out.println(it.next());
  }
}
```

Ejemplo con otra estructura

Pregunta

¿cuáles serían los cambios sobre el método `show` para recorrer una estructura de datos de tipo `FIFO` o `LIFO`?

```
public static <E> void show(FIFO<E> list){
  Iterator<E> it = list.iterator();
  while (it.hasNext()) {
     System.out.println(it.next());
  }
}
```

Ejemplo con otra estructura

Pregunta

¿cuáles serían los cambios sobre el método `show` para recorrer una estructura de datos de tipo `FIFO` o `LIFO`?

```
public static <E> void show(FIFO<E> list){
  Iterator<E> it = list.iterator();
  while (it.hasNext()) {
     System.out.println(it.next());
  }
}
```

- Ahora con un bucle `for`

```
for (Iterator<E> it=list.iterator(); it.hasNext(); ){
  System.out.println(it.next());
}
```

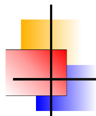
Interfaces Iterator<T> e Iterable<T>

- `java.util.Iterator<E>` declara todos los métodos que debe implementar un objeto iterador

```
public interface Iterator<E> {
  public E next();           /* obligatorio */
  public boolean hasNext();  /* obligatorio */
  public void remove();      /* problematico */
}
```

- `java.util.Iterable<E>` es la pieza que nos permitirá asociar iteradores a un TAD

```
public interface Iterable<E> {
  public Iterator<E> iterator();
}
```



Métodos Iterator

- `hasNext` indica si el cursor referencia a un elemento (es distinto de `null`)
  - **NO** debe entenderse como *¿hay siguiente cursor?*
- `next` guarda el elemento al que apunta el cursor, avanza el cursor y devuelve el elemento guardado
  - **NO** debe interpretarse como *dame el siguiente elemento*
  - Puede dejar el cursor a `null` después de avanzar (si es el último)
- `remove` borra el elemento que devolvió `next` en su última ejecución
  - Es necesario que previamente se haya ejecutado `next`
  - No es obligatorio implementarlo en la asignatura

Ejemplo Iterador

```
Iterator<E> = tad.iterator();

 {A,B}     hasNext() devuelve true
  ↑        next() avanza el cursor a B y devuelve A
cursor
```

```
Iterator<E> = tad.iterator();

 {A,B}     hasNext() devuelve true
  ↑        next() avanza el cursor a B y devuelve A
cursor

 {A,B}     hasNext() devuelve true
    ↑      next() avanza el cursor a null y devuelve B
  cursor
```

Ejemplo Iterador

```
Iterator<E> = tad.iterator();

 {A,B}     hasNext() devuelve true
  ↑        next() avanza el cursor a B y devuelve A
cursor

 {A,B}     hasNext() devuelve true
    ↑      next() avanza el cursor a null y devuelve B
  cursor

 {A,B}     hasNext() devuelve false
           next()    lanza NoSuchElementException
 cursor
   ↓
  null
```

Objeto Iterador

- Si al crearse el iterador el TAD está **vacío**
  - `hasNext` devuelve `false`
  - `next` lanza `NoSuchElementException`
- Si el TAD **no** está **vacío**
  - `hasNext` devuelve `true`
  - `next` avanza el cursor devolviendo el elemento actual
- Si el cursor apunta a `null`
  - `hasNext` devuelve `false`
  - `next` lanza `NoSuchElementException`

Pregunta

¿es posible usar a la vez dos iteradores sobre el mismo TAD?

Método remove

- Debe borrar del TAD el elemento devuelto por el último `next`
- Si no ha habido `next` → `IllegalStateException`

```
while (it.hasNext()) {
  it.next();
  it.remove();     // correcto
}

if (!it.hasNext())
  it.remove();     // Incorrecto, no hay elementos

if (it.hasNext())
  it.remove();     // Incorrecto, no hay next previo

it.next();         // correcto
it.remove();       // correcto
```

Ejercicio

Método que devuelve el décimo elemento de una lista

Ejemplo: Décimo

Ejercicio

Método que devuelve el décimo elemento de una lista

```
public static <E> E tenth(PositionList<E> list)
                throws NoSuchElementException {
  if (list.size() < 10) {
    throw new NoSuchElementException();
  }
  Iterator<E> it = list.iterator();
  for (int i = 1; i < 10; i++) {
    it.next();
  }
  return it.next();
}
```

Ejercicio

Método que borra todos los elementos de una lista

Ejemplo: Borra todos

Ejercicio

Método que borra todos los elementos de una lista

```
public <E> void deleteAll(PositionList<E> list) {
  Iterator<E> it = list.iterator();
  while (it.hasNext()) {
    it.next();
    it.remove();
  }
}
```

NOTA!!

Recordad que `remove` debe invocarse siempre después de `next`

Ejemplo: Suma

Ejercicio

Método que devuelve la suma de los elementos de una lista de enteros

Ejemplo: Suma

Ejercicio

Método que devuelve la suma de los elementos de una lista de enteros

```
public int sumaElems(PositionList<Integer> list) {
  Iterator<E> it = list.iterator();
  int suma = 0;
  while (it.hasNext())
    suma += it.next(); // Asumimos != null
  return suma;
}
```

Ejemplo: Sublista

Ejercicio

Método que indica si una lista es sublista de otra

Ejemplo: Sublista

Ejercicio

Método que indica si una lista es sublista de otra

```
public static <E> boolean sublist(PositionList<E> l1,
    PositionList<E> l2) {
  if (l1 == null || l2 == null) return false;
  if (l1 == l2) return true;
  Iterator<E> it1 = l1.iterator();
  boolean res = false;
  while ( it1.hasNext() &&
          (res = this.member(it1.next(),l2)) )
     ;
  return res;
}
```

Ejemplo: Sublista

Ejercicio

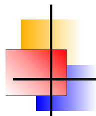
Método que indica si dos listas son iguales

Ejemplo: Sublista

Ejercicio

Método que indica si dos listas son iguales

```
<E> boolean iguales (PositionList<E> list1,
                     PositionList<E> list2) {
    ...
    Iterator<E> it1 = list1.iterator();
    Iterator<E> it2 = list2.iterator();
    E e1, e2;
    boolean iguales = true;
    while (it1.hasNext() && iguales) {
      e1      = it1.next();
      e2      = it2.next();
      iguales = igualesElem(e1,e2);
    }
    return iguales;
  }
```



¿Cómo y cuándo usar iteradores?

- Los iteradores se usan para iterar sobre TADs que son colecciones de elementos
  - No todos los TADs serán colecciones de elementos
  - Hay TADs que son colecciones de elementos pero dada su naturaleza no son iterables
- El problema debe requerir únicamente el acceso a elementos (`next`)
  - No permite el acceso al nodo (sólo al elemento)
- Sólo se puede borrar el último elemento devuelto por el iterador (`remove`)

Implementación de Iteradores: Alternativas

- El objeto iterador itera **usando los métodos del interfaz del TAD**
  - El iterador puede usarse para iterar sobre objetos de cualquier clase que implemente el interfaz
  - El iterador puede iterar sobre cualquier clase que implemente 'I' si usa únicamente métodos de 'I' para "mover" el cursor.
- El objeto mueve el cursor **accediendo de los atributos de la clase que implementa el TAD**
  - Únicamente pueden usarse para iterar sobre objetos de las clases concretas
  - Si 'C' que implementa el interfaz 'I', el iterador definido para objetos de 'C' sólo puede usarse sobre objetos de 'C'

Pregunta

¿qué ventajas e inconvenientes tiene cada opción?

Iteradores sobre Interfaces (1)

(1) El interfaz del TAD debe extender `Iterable`

- Debe implementar el método `iterator()`

```
import java.util.Iterator;

public interface TAD<E> extends Iterable<E> {
  ... /* metodos del TAD */
  public Iterator<E> iterator() ;
  public Iterable<E> snapshot() ; /* no obligatorio */
}
```

- `snapshot` no forma parte del interfaz `Iterable`
  - Devuelve un iterable con todos los elementos del TAD
  - Permite recorrer el iterador al mismo tiempo que modificar el TAD original

Iteradores sobre Interfaces (2)

(2) Se implementa una clase iterador que usa los métodos del interfaz del TAD (no de la clase) para mover el cursor

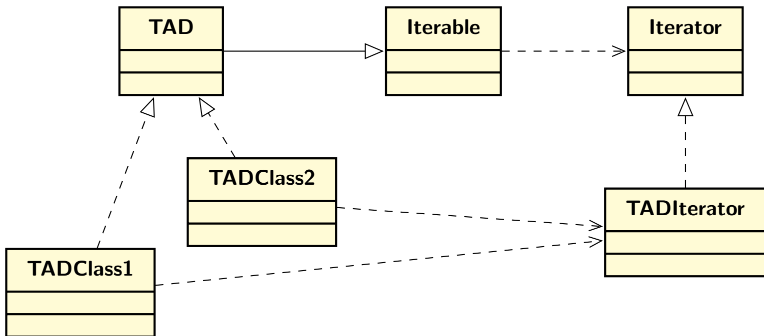
```
import java.util.Iterator;
public class TADIterator<E> implements Iterator<E> {
  TAD<E> tad;  /* el TAD es un atributo */
  CursorTAD<E> cursor;

  public TADIterator(TAD<E> t) {
    tad = t;
    ... /* inicializa el valor del cursor */
  }
  public boolean hasNext()  { /* codigo aqui */ }
  public E next()           { /* codigo aqui */ }
  public void remove()      { /* codigo aqui */ }
}
```

Iteradores sobre Interfaces (3)

(3) Implementar el método `iterator` en todas las clases que implementan el interfaz `TAD<E>`

```
import java.util.Iterator;
public class TADClass1<E> implements TAD<E> {
  ...
  public Iterator<E> iterator() {
   return new TADIterator<E>(this);
  }
}
...
public class TADClass2<E> implements TAD<E> {
  ...
  public Iterator<E> iterator() {
   return new TADIterator<E>(this);
  }
}
```

(4) El diagrama de clases quedaría



Ejemplo

Ver el código de `PositionListIterator<E>`

El bucle for-each

- El bucle `for-each` es una abstracción que simplifica el código del bucle `for` en algunos casos
- Pasamos de este código ...

```
public static <E> void show(PositionList<E> list) {
  Iterator<E> it = list.iterator();
  while (it.hasNext())
    System.out.println(it.next());
}
```

- a este otro código ...

```
public static <E> void show(PositionList<E> list) {
  for (E e : list)
    System.out.println(e);
}
```

- *"para todo elemento `e` de tipo `E` en el iterable `list`"*

Sintaxis de for-each

- Patrón de sintaxis:

```
for (type elem : expr) {
    stmts
}
```

- La variable `elem` tiene tipo `type` y no aparece en `expr`
- La expresión `expr` tiene tipo `Iterable<T>` o tipo "array de T", con T un subtipo de `type`
- Dentro de `stmts` no se tiene acceso al iterador (a una variable que referencie el objeto iterador), sólo al elemento `elem`
- Se recorre el TAD iterable por completo *for-each* = *para cada elemento*
  - **NO** debe utilizarse para recorridos parciales

```
for (Iterator<E> it=list.iterator(); it.hasNext(); ) {
    E e = it.next();
}
```

Ejemplo for-each

Ejemplo

Método toString de la clase PositionList

```
public String toString() {
  String s = "[";
  for (E e : this)) {
    if (e == null)
      s += "null";
    else
      s += e.toString();
    if (cursor != last())
      s += ", ";
  }
  s += "]";
  return s;
}
```

Ejemplo: Suma elementos

Ejemplo

Suma de elementos de una lista de enteros

```
int sumaElems(PositionList<Integer> list) {
  int suma = 0;
  for (Integer e : list)
    if (e != null)
      suma += e;
  return suma;
}
```

For-each en Arrays

- `for-each` también se puede utilizar para recorrer arrays

Ejemplo

Suma de los elementos de un array

```
public int sumaArray(int [] v) {
  int suma = 0;
  for (int e : v)
    suma += e;
  return suma;
}
```