

Uso della classe `StringList`: aggiungere e togliere elementi

- Vogliamo eliminare tutti gli elementi iniziati per "E".
Provate: nessuno dei due schemi precedenti funziona!
- Occorre usare uno schema ancora leggermente diverso: **realizzatelo per esercizio** (ricordando che per poter eliminare un nodo occorre avere un riferimento al precedente).
- I metodi della classe `StringList` permettono all'utilizzatore di creare un cursore con cui scorrere la lista e modificarla, anche aggiungendo e togliendo nodi, e garantiscono il mantenimento dell'invariante. Tuttavia non permettono di adottare uno schema uniforme di ciclo per inserimenti, cancellazioni, ecc.
- Per permettere uno schema uniforme è conveniente definire un oggetto cursore più complesso, contenente due riferimenti a due nodi consecutivi, in modo da poter trattare più semplicemente il caso della cancellazione;
- nella terminologia Java: un oggetto **iteratore**.

Senza definire una classe `Iterator`: soluzione insoddisfacente

- Si definiscono nella classe `List` due altri campi interni di tipo `Node`, *corrente* e *precedente*, con metodi per:
 - inizializzarli (impostando *corrente* al primo elemento);
 - avanzare di un passo;
 - sapere se la lista ha ancora degli elementi oppure è finita
 - modificare o cancellare l'elemento corrente;
 - ecc.
- La classe annidata `Node` può essere resa privata, quindi invisibile all'utilizzatore, poiché non serve più.
- Svantaggi: così si ha un solo cursore per la lista, e non è possibile percorrere la lista contemporaneamente con due cursori, come ad esempio nel selection-sort o nell'insertion sort.

Nota terminologica

Nello scrivere e parlare di programmazione a oggetti si usa spesso, per brevità, la locuzione:

un oggetto *NomeDiClasse*

invece di:

un oggetto (o istanza) della classe *NomeDiClasse*.

Esempio:

un oggetto `StringList` = un oggetto della classe `StringList`

Non si confondano però gli oggetti con le loro classi!
Nella programmazione Java avanzata (che non faremo), si vede che anche ogni classe è rappresentata da un oggetto (della classe `Class` ...); naturalmente, non si deve confondere l'oggetto-classe con gli oggetti della classe!

Gli iteratori: introduzione.

- ogni oggetto della classe `StringListIterator` rappresenta uno stato astratto di "percorrenza di una lista", costituito da:
 - indicazione dell'**ultimo nodo visitato** (o nodo **corrente**);
 - indicazione del **penultimo visitato** (o nodo **precedente**);
- su tale stato astratto sono definiti i seguenti predicati:
 - **ci sono ancora elementi da visitare**;
 - **è permesso effettuare una cancellazione**;
- ogni invocazione del metodo **next** fa due azioni:
 - avanza al nodo successivo;
 - restituisce il dato contenuto in tale nodo successivo;

Gli iteratori: introduzione (continua).

- per percorrere le liste vogliamo usare oggetti (della classe) `StringListIterator` **invece di** oggetti della classe `Node`, la quale sarà pertanto dichiarata **solo package-visibile**.
- ma quando si crea un oggetto `StringListIterator`, bisogna passargli il nodo da cui inizia l'iterazione:
se la classe-nodo non è pubblica, come si fa?

due possibili soluzioni:

1. si passa come argomento al costruttore dell'iteratore l'oggetto-lista su cui operare, e si lascia il campo `first` della classe-lista visibile nel package;
2. gli oggetti iteratori vengono creati e forniti all'utilizzatore dagli oggetti-lista stessi, e solo da essi, attraverso opportuni metodi della classe-lista (soluzione SUN).

Gli iteratori: introduzione (continua).

- per percorrere e modificare una lista ho bisogno dei metodi `next()`, `addAfter(...)`, `removeNext()`; questi metodi erano nella classe `Node`,
ma ora `Node` non è pubblica: come si fa?
- se il dato contenuto nel nodo è di tipo primitivo, per poterlo modificare occorre che il campo-dati sia pubblico;
- se il dato nel nodo è un (riferimento a) oggetto, ma si vuole poter sostituire l'intero oggetto, il campo-dati deve essere pubblico;
ma ora `Node` non è pubblica: come si fa?

ovvia soluzione:

- `next()`, `addAfter(...)`, `removeNext()` e un nuovo metodo `set()` diventano metodi della classe-iteratore:
- o definiti direttamente (e solamente) nell'iteratore;
 - oppure richiamanti gli omonimi metodi di `Node`;

Soluzione 1: iteratore creato all'esterno della lista

```
package lists;

public class StrListIterator {
    private Node current;
    private Node previous;

    public StrListIterator(StringList lis) {
        current = lis.first;
        previous = null;
    }
    ...
}
```

```
package lists;

public class StringList {
    ...
    Node first; // visibile nel package
    ...
}
```

Soluzione 1: utilizzazione

```
...
class UseStringList {

    public static void main(String[] args) {
        StringList myList = StringList.read("nomi.txt");

        StrListIterator i = new StrListIterator(myList);
        while(i.hasNext()) {
            String elem = i.next();
            i.set("Ciao, " + elem);
        }
        // supponendo di aver definito in StringList il metodo toString
        System.out.println(myList);
    }
}
```

Soluzione 1: osservazioni

- se gli oggetti-iteratori non conservano al loro interno il riferimento al primo elemento della lista su cui operano, ognuno di essi può essere usato una sola volta per percorrere la lista;
- se si vuole che gli iteratori possano essere "resettati" a puntare di nuovo all'inizio della lista, basta aggiungere alla classe-iteratore un campo **first** in cui viene inizialmente copiato il **first della lista argomento**, oltre naturalmente ad un metodo **reset**.

Soluzione 2: iteratore creato (solo) all'interno della lista

```
package lists;

public class StrListIterator {
    private Node current;
    private Node previous;

    costruttore non pubblico !
    StrListIterator(StringList lis) {
        current = lis.first;
        previous = null;
    }
}
```

```
package lists;

public class StringList {
    ...
    public StrListIterator iterator() {
        return new StrListIterator(this);
    }
}
```

Soluzione migliore: iteratore come classe interna

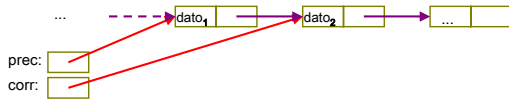
- si definisce la classe-iteratore **dentro** la classe-lista, ma definendola come **non statica**, cioè come vera **inner class**;
- così (vedi pg. 23) un oggetto della classe-iteratore ha un "**oggetto circondante**" della classe lista, e può accedere direttamente ai campi di tale oggetto;
- la **classe-iteratore** può essere addirittura resa **privata**, purché di definisca un'interfaccia-iteratore (pubblica) e si dichiari la classe interna privata iteratore come implementante l'interfaccia;
- in questo modo l'**utilizzatore** non deve preoccuparsi del nome composto *ClasseLista.ClasseIteratore*: anzi, **non conosce il nome della classe degli iteratori** che riceve e usa! conosce solo il nome dell'interfaccia;
- è la soluzione adottata dalla SUN.

Iteratore: eliminazione del nodo corrente.

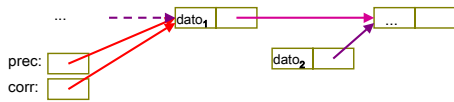
- La conoscenza del nodo precedente al corrente permette la cancellazione del nodo corrente;
- affinché in un ciclo l'avanzamento di un nodo faccia sempre passare al primo nodo ancora da visitare (senza saltarne nessuno), occorre che **dopo la cancellazione si consideri come nuovo nodo corrente il precedente**;
- ma allora non è possibile aggiornare correttamente il precedente, che dovrebbe diventare il precedente del precedente; lasciandolo invariato, esso risulta invece uguale al corrente;
- in questa situazione quindi non sarà quindi possibile cancellare il corrente, perché non abbiamo il suo vero precedente;
- la situazione di "**non è permessa la cancellazione**" è quindi segnalata dalla **coincidenza "anomala" fra corrente e precedente**; non c'è pertanto bisogno di una rappresentazione esplicita di tale stato per mezzo di un campo.
- in questo stato "anomalo" il primo avanzamento del nodo corrente riporta il precedente ad essere il vero precedente, e quindi rende di nuovo possibile una successiva cancellazione.

Eliminazione del nodo corrente

PRIMA



DOPO



La classe StringListIterator: specifica

Definizione preliminare di due nozioni:

- **nodo corrente o ultimo-visitato**: è il nodo il cui dato è stato restituito dall'ultima invocazione del metodo `next()`; se `next()` non è mai stato invocato, è nullo;
- **nodo precedente o penultimo**: se è permesso effettuare una cancellazione se il nodo corrente è nullo o è il primo, è nullo

Metodi (pubblici):

- **boolean hasNext()**: è true se il successivo del corrente non è nullo; nel caso di corrente nullo, è true se il primo della lista è non nullo;
- **boolean next()**:
PRE : `hasNext()` è true, corrente = C, precedente = P
POST: C non è nullo $\Rightarrow$ corrente = successivo di C e precedente = C;
C è nullo $\Rightarrow$ corrente = primo e precedente = nullo;
"è permesso effettuare una cancellazione" è vero;

La classe StringListIterator: specifica (continua)

- **void add(String elem)**: aggiunge un elemento dopo il corrente o, se il corrente è nullo, lo aggiunge in testa; l'elemento aggiunto diventa il nuovo corrente, il vecchio corrente diventa il precedente;
"è permesso effettuare una cancellazione" è vero;
(per esercizio si scrivano più precisamente PRE e POST)
- **void remove()**:
PRE: "è permesso effettuare una cancellazione" è vero;
elimina il nodo corrente;
se il nodo corrente è il primo, aggiorna il riferimento al primo;
il nuovo nodo corrente è il precedente;
"è permesso effettuare una cancellazione" diventa falso
(per esercizio si scrivano più precisamente PRE e POST)
- **public void set(String element)**: cambia l'ultimo elemento visitato.

Un'interfaccia iteratore

```
package liste;

public interface StringListIterator
extends java.util.Iterator<String> {
    boolean hasNext();
    String next();
    void add(String element);
    void remove();
    void set(String element);
}
```

Ricorda: i metodi dichiarati in una interfaccia sono implicitamente pubblici.

La classe StringList con il metodo che fornisce iteratori

```
public class StringList2 implements Iterable<String> {
    private static class Node {
        ...
    }

    private class Itr implements StringListIterator {
        private Node current;
        private Node previous;
        ...
    }

    private Node first;
    ...
    public StringListIterator iterator() {
        return new Itr();
    }
    ...
}
```

Utilizzazione

```
StringList2 myList =
    StringList2.read("nomi.txt");

StringListIterator i = myList.iterator();
while(i.hasNext()) {
    if(i.next().startsWith("E")) i.remove();
}

eccetera.
```