Programmazione in Java
Parte II

Lezione 7

Dott. Marco Faella

- Java Collection Framework (JCF) è una parte della libreria standard dedicata alle *collezioni*, intese come classi deputate a contenere altri oggetti
- Questa libreria offre strutture dati di supporto, molto utili alla programmazione, come liste, array di dimensione dinamica, insiemi, mappe associative (anche chiamate *dizionari*) e code

- Le classi e interfacce del JCF si dividono in due gerarchie:
  - quella che si diparte dall'interfaccia Collection
  - e quella che si diparte da Map

- Inoltre, la classe Collections (si noti la “s” finale) contiene numerosi algoritmi di supporto
  - ad esempio, metodi che effettuano l'ordinamento

- La classe java.util.LinkedList<E> rappresenta una lista doppiamente concatenata
- La classe ha un **parametro di tipo** che indica il tipo degli oggetti contenuti
- Essa appartiene alla Java Collection Framework
- Qui, presentiamo brevemente i suoi metodi principali

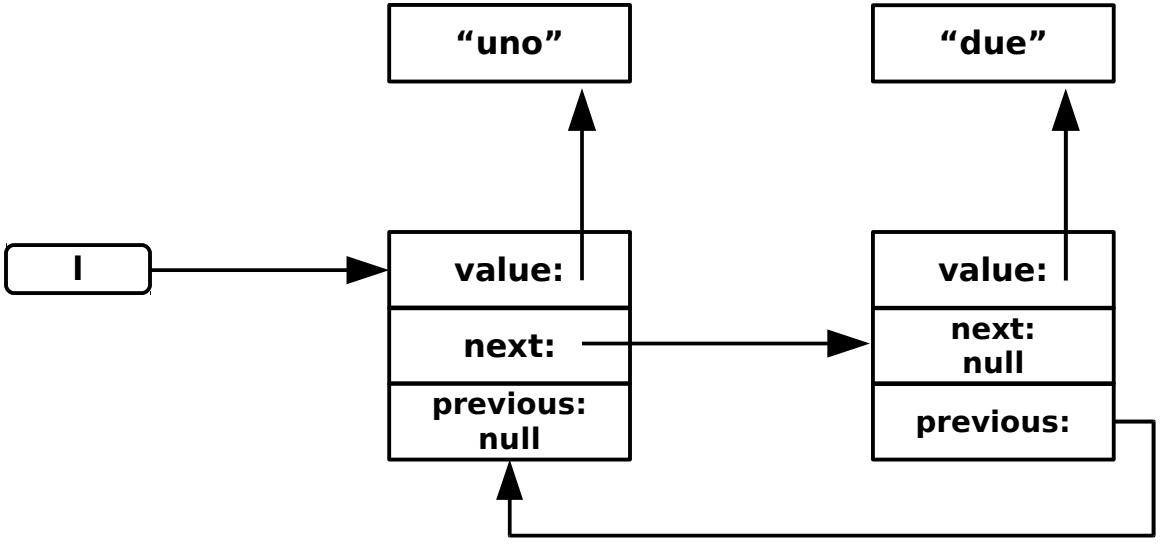
public boolean add(E x)	**aggiunge x in coda alla lista e restituisce true**
public boolean contains(Object x)	**restituisce true se la lista contiene l'oggetto x; ha complessità proporzionale alla lunghezza della lista**
public int size()	**restituisce la dimensione della lista**
public void addFirst(E x)	**aggiunge x in testa alla lista**
public void addLast(E x)	**equivalente ad add(x), ma senza valore restituito**
public E removeFirst()	**rimuove e restituisce la testa della lista**
public E removeLast()	**rimuove e restituisce la coda della lista**

Esempio:

```
LinkedList<String> l = new LinkedList<String>();

l.add(“uno”);
l.add(“due”);
```

Memory layout:



Il ciclo for-each

- Java 1.5 ha introdotto un nuovo tipo di ciclo for, chiamato ciclo “for each”
- Ad esempio:

```
String[] array = {“uno”, “due”, “tre”};
for (String s: array)
   System.out.println(s);
```

- Il ciclo di sopra stampa tutte le stringhe contenute nell'array, una per rigo
- Questa nuova forma di ciclo permette di iterare su un array senza dover esplicitamente utilizzare un indice
  - quindi senza il rischio di sbagliare gli estremi dell'iterazione

- Oltre che per gli array, il ciclo for-each funziona anche sulle liste
  - precisamente, il ciclo for-each funziona su tutti gli oggetti che implementano l'interfaccia Iterable<E>

- Esercizio: scorrere una lista

L'interfaccia List<E>

- LinkedList<E> implementa l'interfaccia List<E>
- Anche ArrayList<E> implementa List<E>
- I metodi principali dell'interfaccia sono i seguenti

`int size()`	già visto in LinkedList
`boolean isEmpty()`	restituisce vero se e solo se la lista è vuota
`boolean add(E x)`	già visto in LinkedList
`boolean contains(Object x)`	già visto in LinkedList
`boolean remove(Object x)`	rimuove l'oggetto x dalla lista; restituisce vero se e solo se un tale elemento era presente

Accesso posizionale

- Inoltre, sono presenti i due metodi responsabili per l'**accesso posizionale**
- **get**(int i) restituisce l'elemento al posto i-esimo della sequenza; solleva un'eccezione se l'indice è minore di zero o maggiore o uguale di size()
- **set**(int i, E elem) sostituisce l'elemento al posto i-esimo della sequenza con elem; restituisce l'elemento sostituito; come get, solleva un'eccezione se l'indice è scorretto

Utilizzare LinkedList come stack o come coda

- Questi metodi permettono di utilizzare una LinkedList sia come **stack** sia come **coda**
- Per ottenere il comportamento di uno **stack** (detto **LIFO**: last in first out), inseriremo ed estrarremo gli elementi dalla **stessa estremità** della lista
    - ad esempio, inserendo con con addLast (o con add) ed estraendo con removeLast
- Per ottenere, invece, il comportamento di una **coda** (**FIFO**: first in first out), inseriremo ed estrarremo gli elementi da due **estremità opposte**

- ArrayList è un'implementazione di List, realizzata internamente con un array di dimensione dinamica

- Ovvero, quando l'array sottostante è pieno, esso viene riallocato con una dimensione maggiore, e i vecchi dati vengono copiati nel nuovo array
  - questa operazione avviene in modo trasparente per l'utente
  - il metodo size restituisce il numero di elementi effettivamente presenti nella lista, non la dimensione dell'array sottostante

- Il ridimensionamento avviene in modo che l'operazione di inserimento (add) abbia complessità *ammortizzata* costante
  - per ulteriori informazioni sulla complessità ammortizzata, si consulti un testo di algoritmi e strutture dati

Le liste e l'accesso posizionale

- L'accesso posizionale (metodi get e set) si comporta in maniera molto diversa in LinkedList rispetto ad ArrayList

- In LinkedList, ciascuna operazione di accesso posizionale può richiedere un tempo proporzionale alla lunghezza della lista
  - difatti, per accedere all'elemento di posto n è necessario scorrere la lista, a partire dalla testa, o dalla coda, fino a raggiungere la posizione desiderata
- In ArrayList, ogni operazione di accesso posizionale richiede tempo costante

- Pertanto, è fortemente sconsigliato utilizzare l'accesso posizionale su LinkedList
- Se l'applicazione richiede l'accesso posizionale, è opportuno utilizzare un semplice array, oppure la classe ArrayList

Esercizio (esame 19/2/2009)

Si implementi la classe `Container`, che rappresenta un contenitore per liquidi di forma fissata.

Ad un contenitore, inizialmente vuoto, si può aggiungere acqua con il metodo `addWater`, che prende come argomento il numero di litri.

Il metodo `getAmount` restituisce la quantità d'acqua presente nel contenitore.

Il metodo `connect` prende come argomento un altro contenitore, e lo collega a questo con un tubo. Dopo il collegamento, la quantità d'acqua nei due contenitori (e in tutti quelli ad essi collegati) sarà la stessa.

Esempio d'uso (l'output è sulla slide successiva):

```
Container a=new Container(), b=new Container(), c=new Container(), d=new Container();

a.addWater(12);
d.addWater(8);
a.connect(b);
System.out.println(a.getAmount()+" "+b.getAmount()+" "+c.getAmount()+" "+d.getAmount());
b.connect(c);
System.out.println(a.getAmount()+" "+b.getAmount()+" "+c.getAmount()+" "+d.getAmount());
c.connect(d);
System.out.println(a.getAmount()+" "+b.getAmount()+" "+c.getAmount()+" "+d.getAmount());
```

Esercizio (esame 19/2/2009)

Output dell'esempio d'uso:

```
6.0  6.0  0.0  8.0
4.0  4.0  4.0  8.0
5.0  5.0  5.0  5.0
```