Java Collections

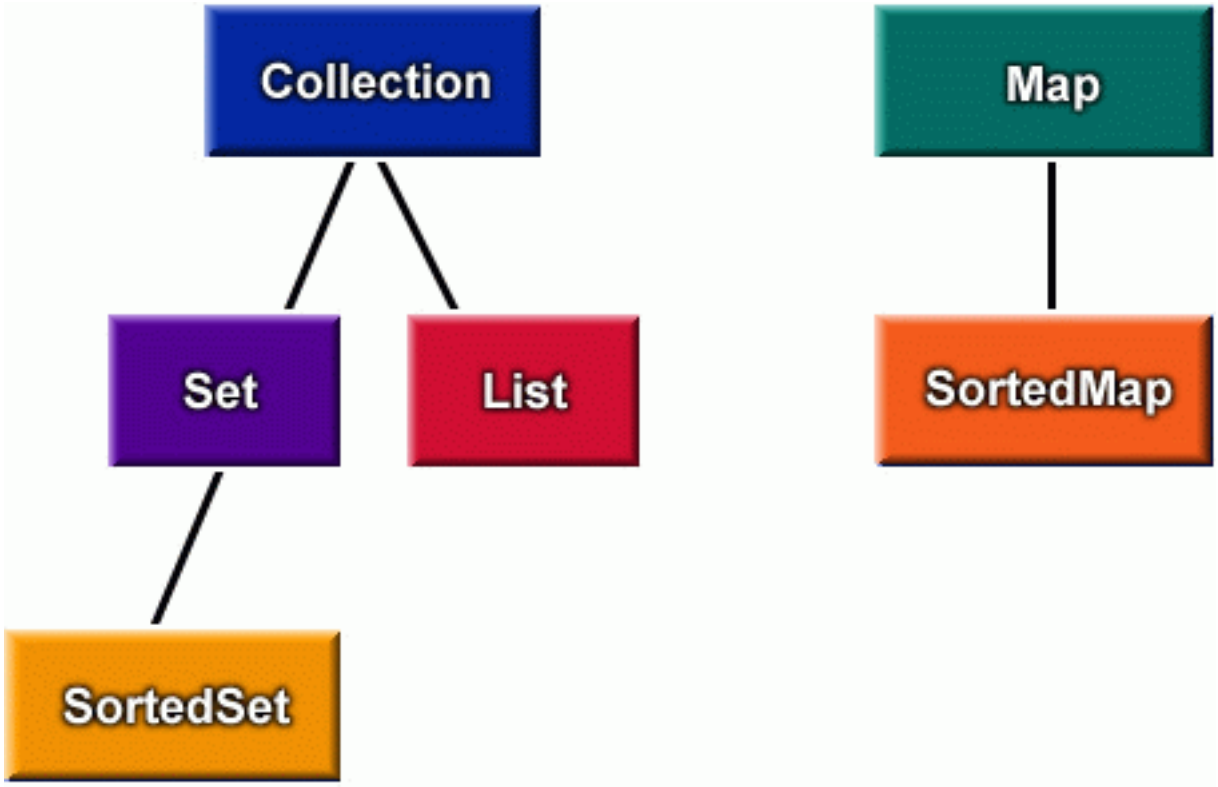
Introduzione

- Una java collection (a volte chiamata anche container) è un oggetto che raggruppa più elementi dello stesso tipo in una singola unità.
- Tipicamente è utilizzata per raggruppare oggetti che nella realtà si trovano spesso insieme (es. i recapiti nella rubrica telefonica, le mail di un casella di posta, etc…)

Metodi di una Collection

```
public interface Collection<E> {
  …
  int size();
  boolean isEmpty();
  boolean contains(E element);
  boolean add(E element);    // Optional
  boolean remove(E element); // Optional
  Iterator iterator();
  …
}
```

Uno sguardo d'insieme



Lista

- Una lista (o sequenza) è una collezione ordinata di oggetti.
- In una lista ogni elemento è caratterizzato da:
  - una posizione
  - un elemento che lo precede
  - un elemento che lo segue
- Una lista può contenere elementi duplicati.

L’interfaccia `List<E>`

Oltre alle operazioni ereditate dall’interfaccia `Collection`, l’interfaccia `List` include operazioni per l’accesso posizionale: gli elementi possono essere manipolati specificando la loro posizione nella sequenza.

List: Accesso posizionale

```
E get(int index);
E set(int index, E element);
void add(int index, E element);
E remove(int    index);
abstract boolean addAll(int index,
  Collection c);
```

List: Search

```
int indexOf(Object o);
int lastIndexOf(Object o);
```

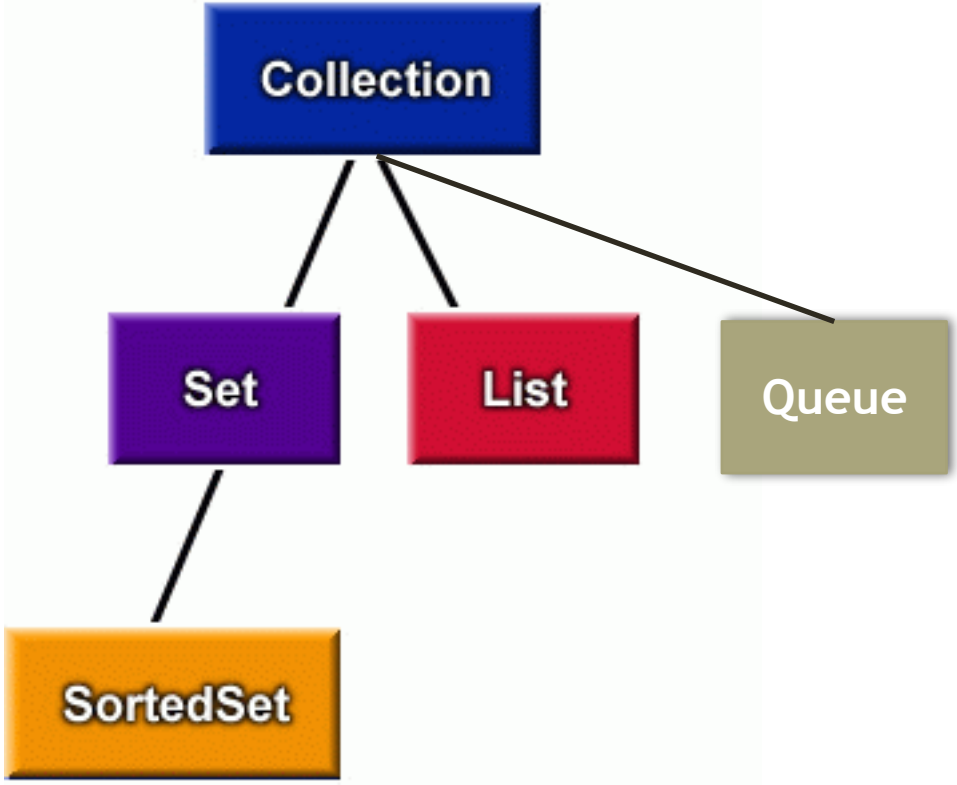
List: Iteration

```
ListIterator listIterator();
ListIterator listIterator(int index);
```

List: Range-view

```
List subList(int from, int to);
```

Uno sguardo d'insieme



Coda

- Una coda è una collezione progettata per mantenere oggetti che devono essere processati.
- Gli elementi in una coda sono mantenuti secondo un certo ordine:
  - FIFO (es. coda alla posta)
  - LIFO (es. stack)
- La testa (o head) della coda è l'elemento che verrà rimosso da una chiamata al metodo `remove()` o `poll()`.

L'interfaccia `Queue<E>`

Oltre alle operazioni ereditate dall'interfaccia `Collection`, l'interfaccia `Queue` include operazioni per

- accedere e/o rimuovere l'elemento che si trova in testa
- aggiungere un elemento in testa o in coda
- Inserire un elemento alla fine (tail) della coda (ordinamento FIFO) o in testa (ordinamento LIFO)

Queue<E>

E element()

Retrieves, but does not remove, the head of this queue

boolean **offer(E o)**

Inserts the specified element into this queue, if possible.

E peek()

Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.

E poll()

Retrieves and removes the head of this queue, or null if this queue is empty.

E remove()

Retrieves and removes the head of this queue.

DALL'INTERFACCIA ALL'IMPLEMENTAZIONE

Ma prima un po’ di ripasso sul concetto di reference

Tipi di dato primitivi

- In una variabile di tipo primitivo, la locazione di memoria identificata dal nome della variabile contiene il valore della variabile.

```
int a;

a = 3;
```

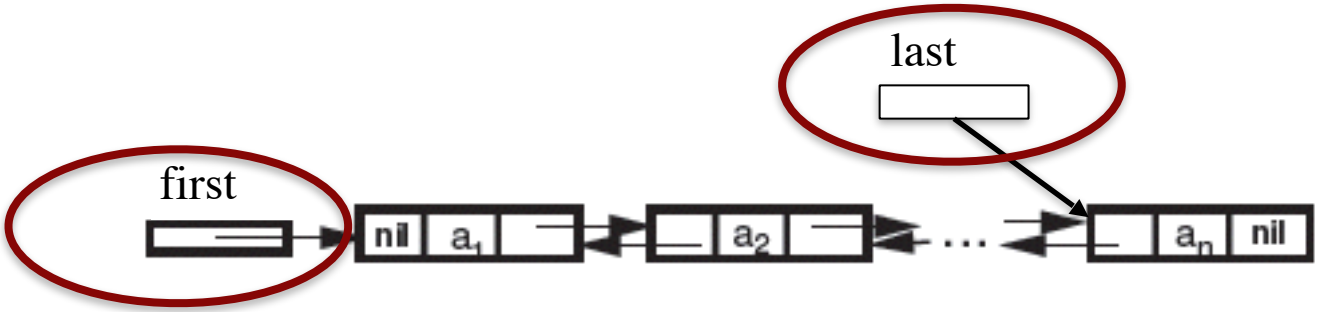
Oggetti

In una variabile di tipo oggetto, la locazione di memoria identificata dal nome della variabile contiene il riferimento (*reference*) ad un'altra locazione di memoria dove è contenuto l'oggetto

```
Integer b;

b = new Integer(3);
```

La classe `LinkedList<T>`



La classe `LinkedList<T>`

- `public class LinkedList<T> …`
  `implements List<T>, Deque<T>, …`

- La LinkedList implementa la struttura dati facendo uso di reference:

```
/**
 * The first element in the list.
 */
Entry<T> first;

/**
 * The last element in the list.
 */
Entry<T> last;
```

Es. Lista bidirezionale

```
private static final class Entry<T>
{
   /** The element in the list. */
   T data;

   /** The next list entry, null if this is last.
   */
   Entry<T> next;

   /** The previous list entry, null if this is
   first. */
   Entry<T> previous;

 }
```

La classe `ArrayList<T>`

public class **ArrayList<E>** ...

implements List<E>, RandomAccess, ...

Implementa le operazioni di una lista utilizzando un array.

Liste vs array

PROS:

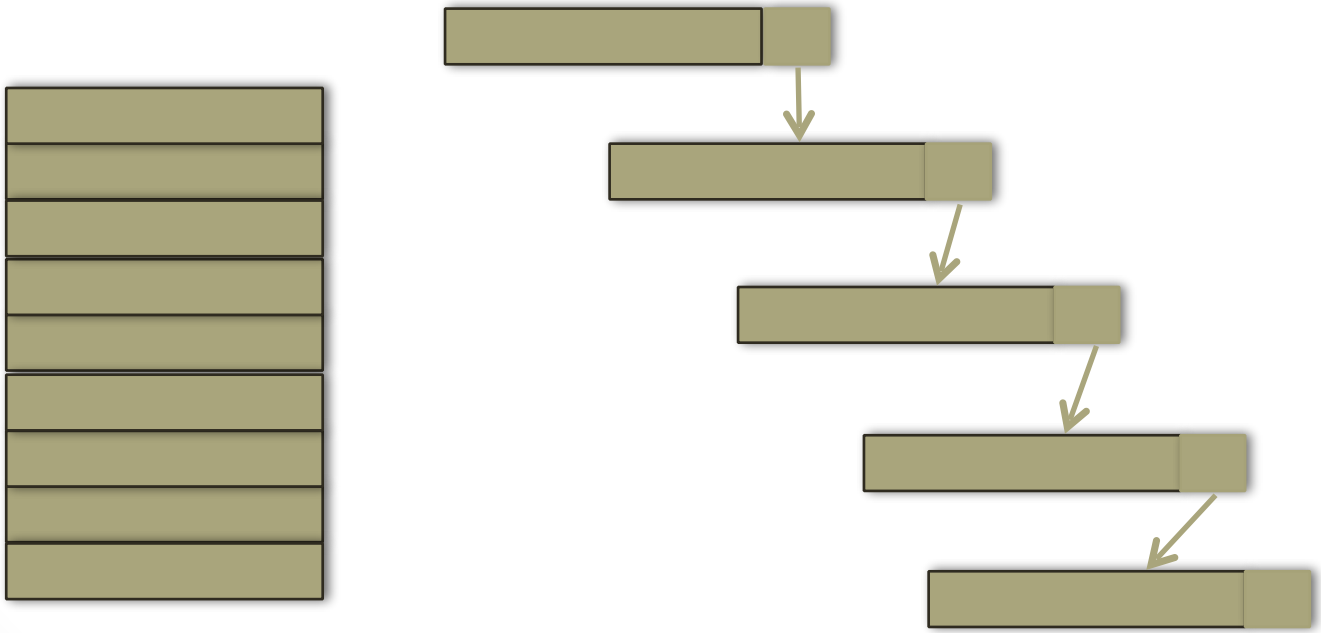
- Minore occupazione di memoria: l'occupazione di memoria è proporzionale al numero di elementi presenti nella struttura dati

CONS:

- Minore efficienza (maggiore complessità computazionale)
- Maggiore complessità di gestione

Efficienza

- In generale, le operazioni effettuate su liste hanno una complessità computazionale maggiore rispetto a quelle effettuate sugli array.
- Negli array è permesso **l'accesso diretto**, nelle liste no



Complessità di gestione

- Occorre gestire i “puntatori” o reference agli altri elementi.
- Java offre delle classi che implementano la gestione puntatori rendendola trasparente agli sviluppatori

Conclusioni

- Differenza tra interfaccia e implementazione
- LinkedList vs. ArrayList