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();
  …
}
```

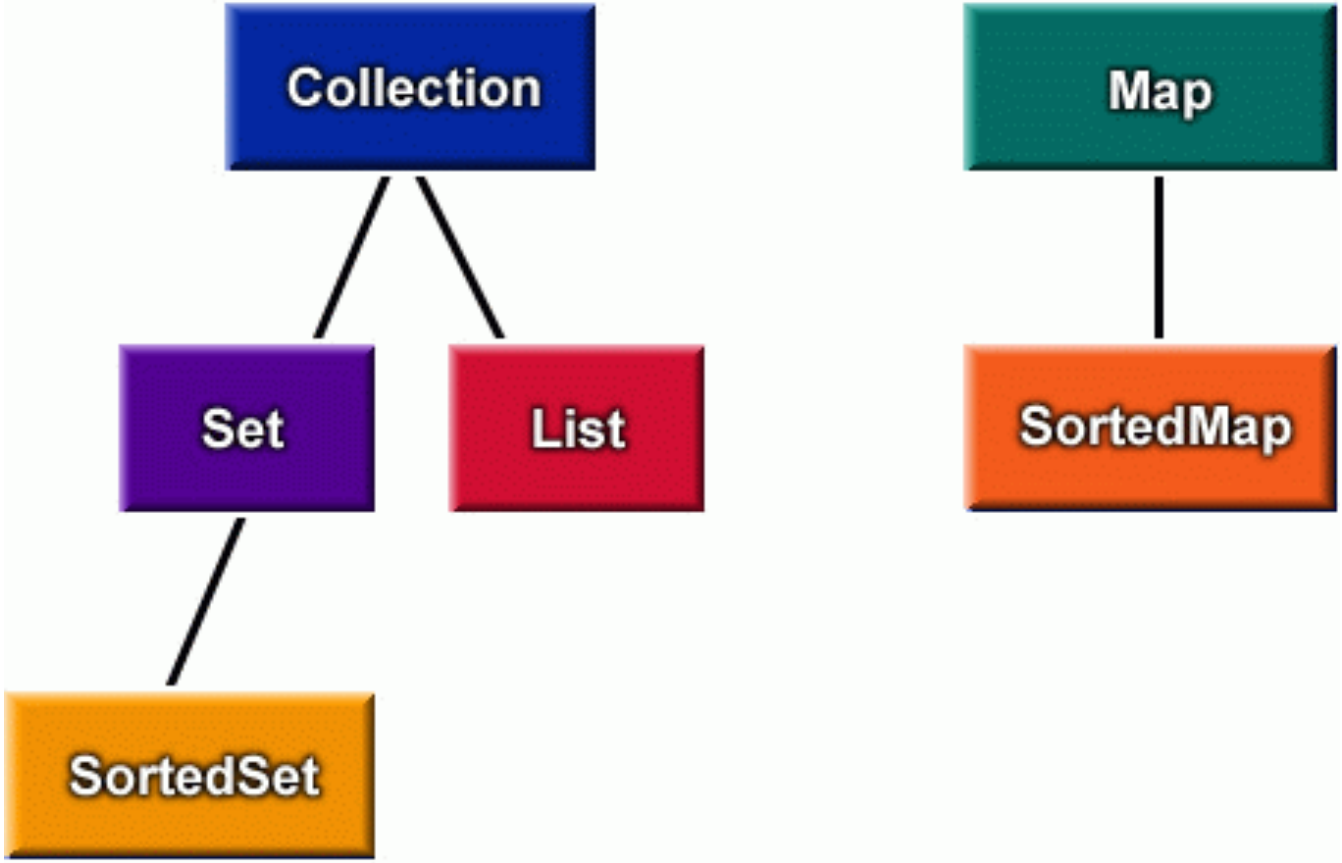
Interface Iterator<E>

```
public interface Collection<E> {

  boolean   hasNext();
  E         next();
  void      remove();

}
```

Uno sguardo d'insieme



Lista

- Una lista (o sequenza) è una collezione ordinata di oggetti.
- In una lista gli 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

```
public interface List<E> extends
  Collection<E> {

    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);
```

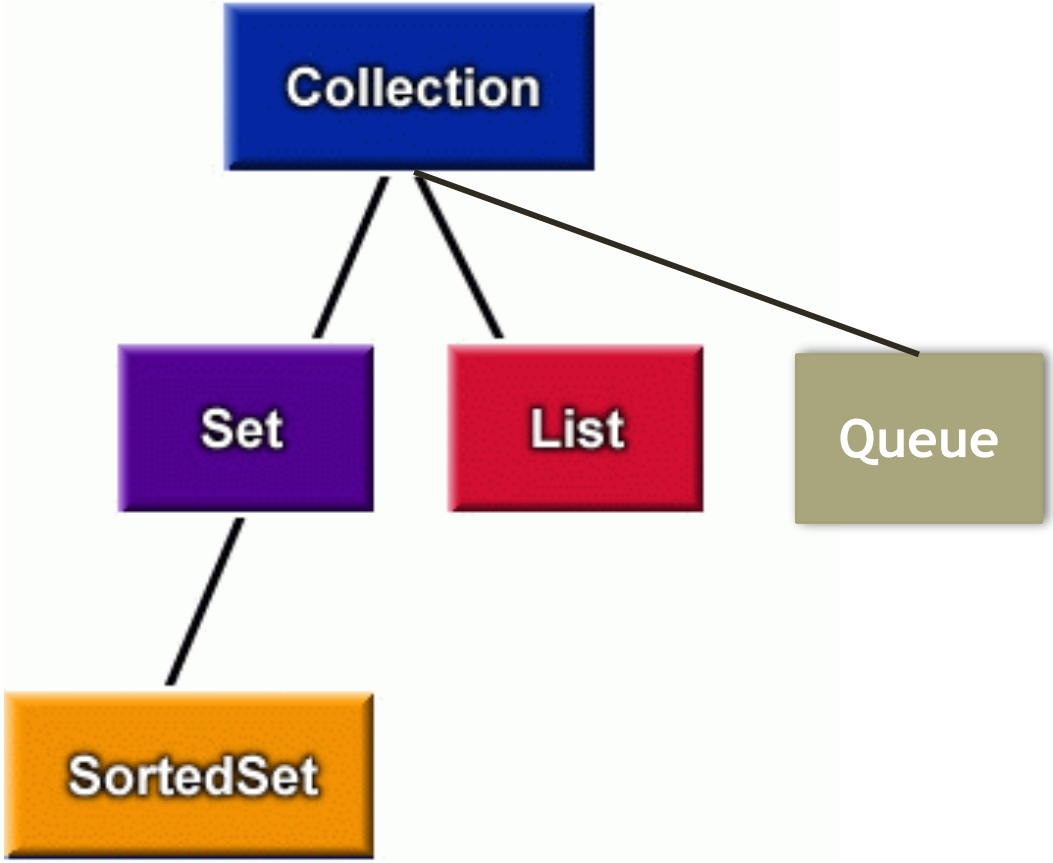
List: Iteration

```
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

```
public interface Queue<E> extends
  Collection<E> {

    boolean add(E e);
    E       element();
    boolean offer(E e);
    E peak();
    E poll();
    E remove();
    …
}
```

L'interfaccia `Queue<E>`

	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

Queue<E>

E element()
Retrieves, but does not remove, the head of this queue.

E peek()
Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.

`boolean offer(E e)`
Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.

`boolean` **`add(E`** `e)`
Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning `true` upon success and throwing an `IllegalStateException` if no space is currently available.

E poll()
Retrieves and removes the head of this queue, or returns `null` if this queue is empty.

E remove()
Retrieves and removes the head of this queue.

DALL'INTERFACCIA ALL'IMPLEMENTAZIONE

La classe `LinkedList<E>`

- `public class LinkedList<E> …`
  `implements List<E>, …`

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

```
int   size;
/**
 * The first element in the list.
 */
Node<E> first;

/**
 * The last element in the list.
 */
Node<E> last;
```

La classe Node<E>

```
private static class Node<E> {
        E item;
        Node<E> next;
        Node<E> prev;

        Node(Node<E> prev, E element, Node<E> next) {
            this.item = element;
            this.next = next;
            this.prev = prev;
        }
    }
```

Il metodo `add(E e)`

```
public boolean add(E e) {
        linkLast(e);
        return true;
    }
```

Il metodo `add(E e)`

```
public boolean add(E e) {
        linkLast(e);
        return true;
    }
```

```
void linkLast(E e) {
        final Node<E> l = last;
        final Node<E> newNode = new Node<>(l, e, null);
        last = newNode;
        if (l == null)
            first = newNode;
        else
            l.next = newNode;
        size++;
        modCount++;
    }
```

Il metodo `indexOf(Object o)`

```
public int indexOf(Object o) {
        int index = 0;
         if (o == null) {
             for (Node<E> x = first; x != null; x = x.next) {
                 if (x.item == null)
                     return index;
                 index++;
             }
         } else {
             for (Node<E> x = first; x != null; x = x.next) {
                 if (o.equals(x.item))
                     return index;
                 index++;
             }
         }
         return -1;
     }
```

Esercizi

- Confronto tra liste e array
- Analizzare e confrontare l'implementazione delle classi LinkedList<T> e ArrayList<T>
- Fornire due implementazioni della classe SquadraDiCalcio, una che fa uso di liste e l'altra che fa uso di array. (Suggerimento: la squadra di calcio può essere vista come una collezione di giocatori)