



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



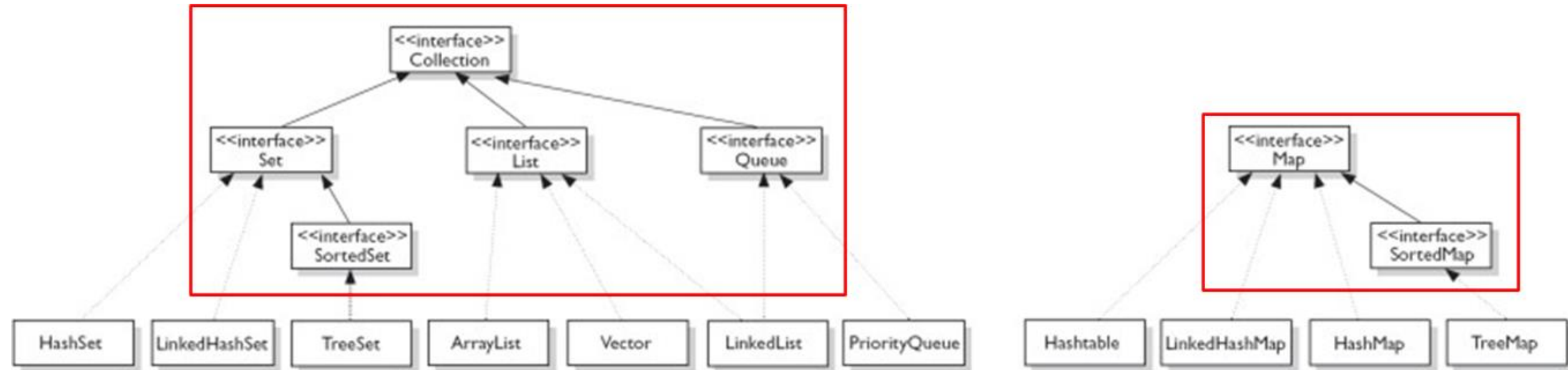
# Laboratorio di Algoritmi e Strutture Dati a.a. 2024/2025

Le classi

HashSet<E> e TreeSet<E>

**Giovanna Melideo**  
Università degli Studi dell'Aquila  
DISIM

# JCF: Interfacce fondamentali (richiami)



# Interfacce fondamentali (richiami)

- **Collection<E>**: nessuna ipotesi sul tipo di collezione. Viene estesa dalle tre interfacce parametriche **Set<E>**, **List<E>** e **Queue<E>**.
- **List<E>**: rappresenta una **sequenza** di elementi, a cui è consentito anche un accesso posizionale
  - sono ammessi duplicati
  - gli elementi al suo interno vengono mantenuti nello stesso ordine in cui sono stati inseriti
- **Queue<E>**: rappresenta una coda di elementi (non necessariamente operante in modo FIFO!)

# Interfaccia Set<E>

- **Set<E>** estende e specializza **Collection<E>** introducendo l'idea di insieme in senso matematico
- La specificità è che un **Set<E>** **non ammette elementi duplicati e non ha una nozione di sequenza o di posizione**
  - **add** aggiunge un elemento solo se esso non è già presente
  - **equals** verifica se due set sono identici:  $|S1|=|S2|, \forall x \in S1, x \in S2$  (e viceversa)
  - tutti i costruttori creano insiemi privi di duplicati

## Method Summary

### Methods

Modifier and Type	Method and Description
boolean	<b>add</b> ( <b>E</b> e) Adds the specified element to this set if it is not already present (optional operation).
boolean	<b>addAll</b> ( <b>Collection</b> <? extends <b>E</b> > c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	<b>clear</b> () Removes all of the elements from this set (optional operation).
boolean	<b>contains</b> ( <b>Object</b> o) Returns <b>true</b> if this set contains the specified element.
boolean	<b>containsAll</b> ( <b>Collection</b> <?> c) Returns <b>true</b> if this set contains all of the elements of the specified collection.
boolean	<b>equals</b> ( <b>Object</b> o) Compares the specified object with this set for equality.
int	<b>hashCode</b> () Returns the hash code value for this set.
boolean	<b>isEmpty</b> () Returns <b>true</b> if this set contains no elements.
<b>Iterator</b> < <b>E</b> >	<b>iterator</b> () Returns an iterator over the elements in this set.
boolean	<b>remove</b> ( <b>Object</b> o) Removes the specified element from this set if it is present (optional operation).
boolean	<b>removeAll</b> ( <b>Collection</b> <?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	<b>retainAll</b> ( <b>Collection</b> <?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	<b>size</b> () Returns the number of elements in this set (its cardinality).
<b>Object</b> []	<b>toArray</b> () Returns an array containing all of the elements in this set.
< <b>T</b> > <b>T</b> []	<b>toArray</b> ( <b>T</b> [] a) Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

# SortedSet<E>

- L'interfaccia **SortedSet<E>** **estende** **Set<E>** e rappresenta un insieme sui cui elementi è definita una relazione d'ordine totale.
- L'iteratore di un SortedSet garantisce che gli elementi saranno visitati in ordine.
- L'interfaccia di accesso aggiunge metodi extra:
  - **first** restituisce l'elemento minimo tra quelli presenti nella collezione;
  - **last** restituisce l'elemento massimo;
  - **headSet**, **tailSet**, **subSet** restituiscono i sottoinsiemi ordinati contenenti gli elementi minori di quello dato, maggiori di quello dato, compresi fra i due dati

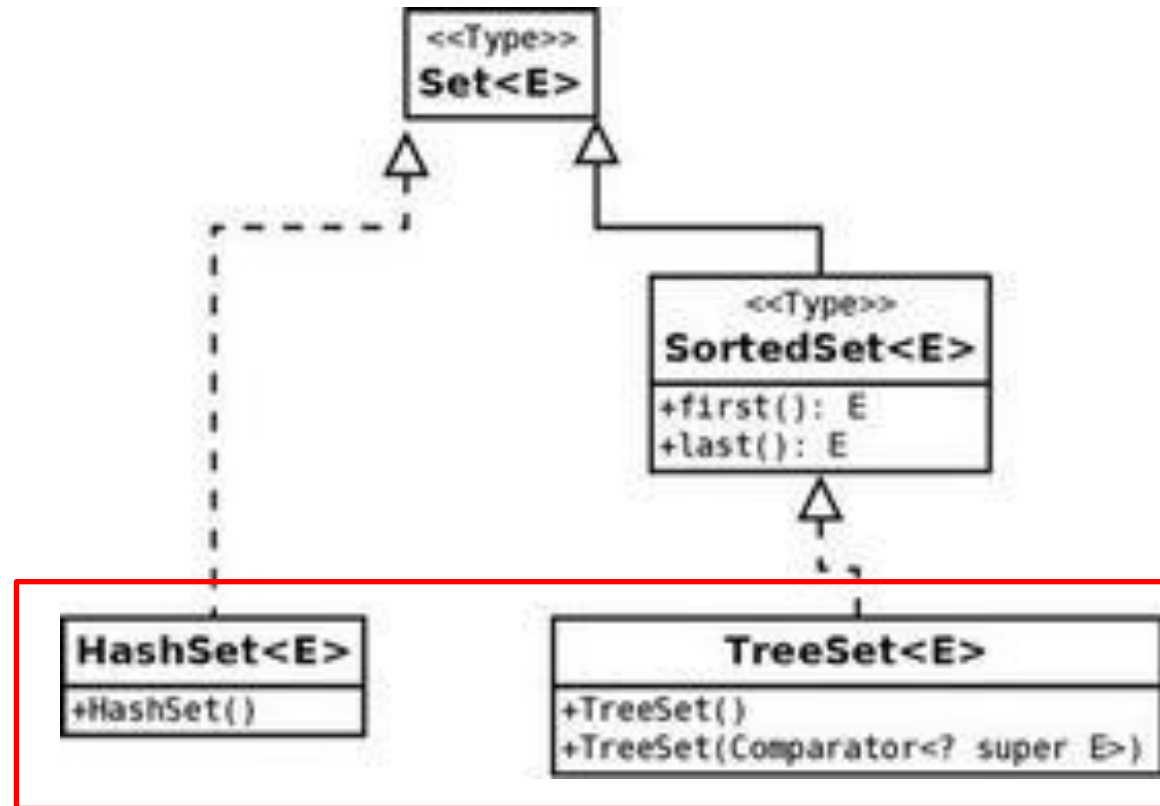
# SortedSet<E>: methods

## Method Summary

### Methods

Modifier and Type	Method and Description
<code>Comparator&lt;? super E&gt;</code>	<code>comparator()</code> Returns the comparator used to order the elements in this set, or <code>null</code> if this set uses the <b>natural ordering</b> of its elements.
<code>E</code>	<code>first()</code> Returns the first (lowest) element currently in this set.
<code>SortedSet&lt;E&gt;</code>	<code>headSet(E toElement)</code> Returns a view of the portion of this set whose elements are strictly less than <code>toElement</code> .
<code>E</code>	<code>last()</code> Returns the last (highest) element currently in this set.
<code>SortedSet&lt;E&gt;</code>	<code>subSet(E fromElement, E toElement)</code> Returns a view of the portion of this set whose elements range from <code>fromElement</code> , inclusive, to <code>toElement</code> , exclusive.
<code>SortedSet&lt;E&gt;</code>	<code>tailSet(E fromElement)</code> Returns a view of the portion of this set whose elements are greater than or equal to <code>fromElement</code> .

# Implementazioni fondamentali





# La classe HashSet<E>

- La classe **HashSet<E>** è l'implementazione di **Set<E>** che viene utilizzata più comunemente
- Utilizza al suo interno una tabella hash
- la tabella hash rappresenta un insieme come un array di liste (buckets)
- usa una **funzione hash** che, dato un elemento, restituisce un numero
  - **public int hashCode()** restituisce un valore intero per un oggetto

# Proprietà di hashCode()

- Se invocato più di una volta sullo stesso oggetto deve ritornare lo stesso intero
  - questo può essere diverso in esecuzioni diverse dell'applicazione; l'importante è che rimanga identico all'interno di una singola esecuzione
- se due oggetti sono uguali secondo il metodo equals() allora hashCode() deve ritornare lo stesso intero
- non è richiesto che a due oggetti diversi (secondo equals()) siano associati due hashCode() diversi

# Esempio: String hashCode() method

## hashCode

```
public int hashCode()
```

Returns a hash code for this string. The hash code for a String object is computed as

$$s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + \dots + s[n-1]$$

using int arithmetic, where  $s[i]$  is the  $i$ th character of the string,  $n$  is the length of the string, and  $^$  indicates exponentiation. (The hash value of the empty string is zero.)

### Overrides:

`hashCode` in class `Object`

### Returns:

a hash code value for this object.

### See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

# La classe HashSet<E>: proprietà

- Le operazioni di inserimento, ricerca e cancellazione possono essere eseguite in tempo costante

## Nota:

- Non c'è un ordine degli elementi
  - non si può dire «inserisci in prima posizione» oppure «trova l'elemento in ultima posizione»
  - l'iteratore non restituisce gli elementi in un ordine particolare
- **Test d'esempio:** `InteriRipetuti.java`

# La classe TreeSet<E>

- TreeSet<E> è un Set implementato internamente come **albero red-black**
- Gli elementi devono essere dotati di una relazione d'ordine, in uno dei seguenti modi:
  - gli elementi devono implementare l'interfaccia **Comparable<>**; in questo caso, si può utilizzare il costruttore di TreeSet senza argomenti;
  - oppure bisogna passare al costruttore di TreeSet un opportuno oggetto **Comparator<>**

# La classe TreeSet<E>: esempi

Esempi di situazioni in cui è conveniente mantenere un insieme ordinato:

- quando si deve frequentemente visualizzare o rappresentare gli elementi dell'insieme in modo ordinato
- quando si devono frequentemente confrontare due insiemi (confrontare due insiemi ordinati è più facile che confrontare due insiemi disordinati)

# La classe TreeSet<E>: costruttori

## Constructor Summary

### Constructors

#### Constructor and Description

`TreeSet()`

Constructs a new, empty tree set, sorted according to the natural ordering of its elements.

`TreeSet(Collection<? extends E> c)`

Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.

`TreeSet(Comparator<? super E> comparator)`

Constructs a new, empty tree set, sorted according to the specified comparator.

`TreeSet(SortedSet<E> s)`

Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.

# La classe TreeSet<E>:main methods

- `int size()` returns the number of elements in this set (its cardinality).
- `boolean isEmpty()` returns true if this set contains no elements.
- `boolean add(E e)` adds the specified element to this set if it is not already present.
- `boolean addAll(Collection<? extends E> c)` adds all of the elements in the specified collection to this set.
- `boolean contains(Object o)` returns true if this set contains the specified element.
- `boolean remove(Object o)` removes the specified element from this set if it is present.
- `void clear()` removes all of the elements from this set.
- `Comparator<? super E> comparator()` returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.
- `E first() / E last()` returns the first (lowest) / the last (highest) element currently in this set.



# La classe TreeSet<E>: complessità

- Le operazioni principali di TreeSet hanno la seguente complessità di tempo, tipica degli alberi di ricerca bilanciati:
  - size  $O(1)$
  - isEmpty  $O(1)$
  - add  $O(\log n)$
  - contains  $O(\log n)$
  - remove  $O(\log n)$
- **Test d'esempio:** `InteriRipetuti.java`
- **Test d'esempio:** `TreeSetExample.java` (ver 1,2,3)

# Set<E>: uso degli iteratori

- L'unico modo di fare cicli per gli insiemi è quello di usare un iteratore
- Quando si crea un iteratore con `it = set.iterator()` e si modifica il set con `set.add` oppure `set.remove`, l'iteratore diventa non più valido (non si può più usare).
- Dopo la modifica del set, non si può più invocare `it.hasNext()` oppure `it.next()`
- **Test d'esempio:** `SetModify.java`



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# Domande?

**Giovanna Melideo**  
Università degli Studi dell'Aquila  
DISIM