

Interfícies

Les interfícies són tipus abstractes de dades (TAD) que defineixen la funcionalitat de les colleccions i funcionalitats de suport. En el framework de colleccions cal distingir-ne dos tipus:

- Interfícies que constitueixen el cor del framework i defineixen els diferents tipus de colleccions: Collection, Set, List, Queue, SortedSet, Map i SortedMap, conegudes normalment com interfícies del JCF.

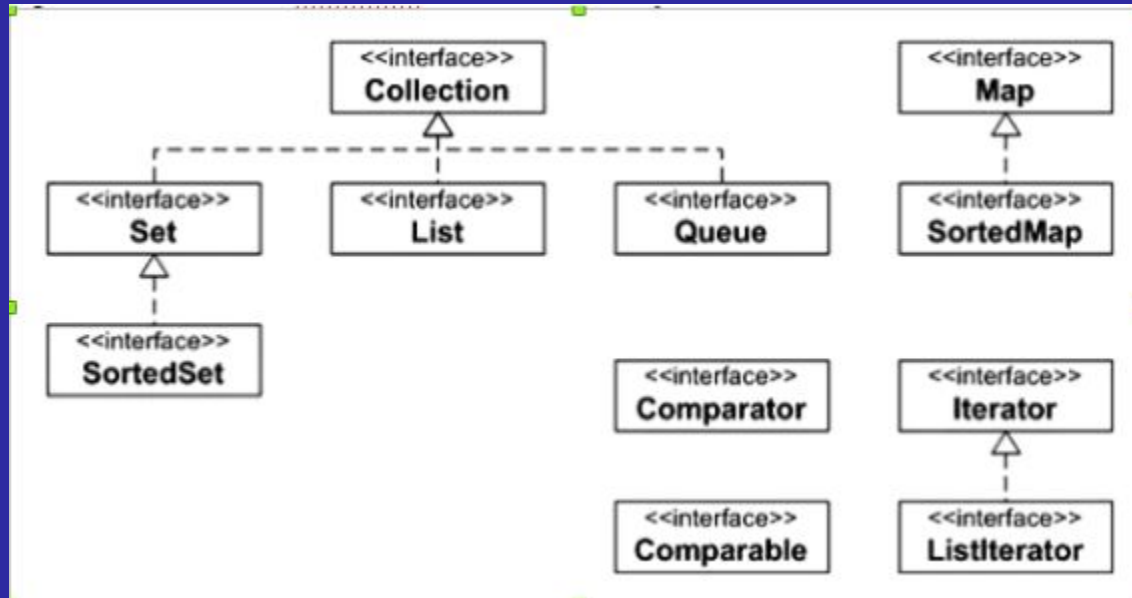
Aquest conjunt d'interfícies, al seu torn, està organitzat en dues jerarquies: una, que agrupa les colleccions amb accés per posició (seqüencial i directe) i que té la interfície Collection per arrel, i l'altra, que defineix les colleccions amb accés per clau i que té la interfície Map per arrel.

- Interfícies de suport: Iterator, ListIterator, Comparable i Comparator.

La figura 1 següent mostra les interfícies del framework de colleccions de Java amb la jerarquia que hi ha entre elles.



Interfícies

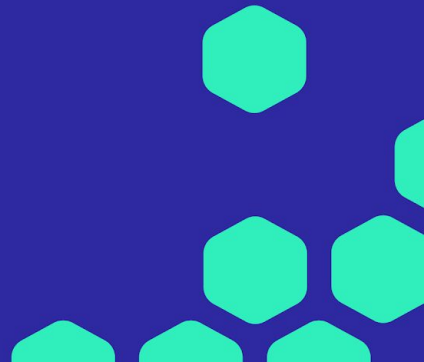


L'explicació detallada de cadascuna de les interfícies del framework de colleccions cal cercar-la en la documentació del llenguatge Java.

Interfície Collection

La interfície `Collection<E>` és la interfície pare de la jerarquia de col·leccions amb accés per posició (seqüencial i directe) i comprèn col·leccions de diversos tipus:

- Col·leccions que permeten elements duplicats i col·leccions que no els permeten.
- Col·leccions ordenades i col·leccions desordenades.
- Col·leccions que permeten el valor null i col·leccions que no el permeten.



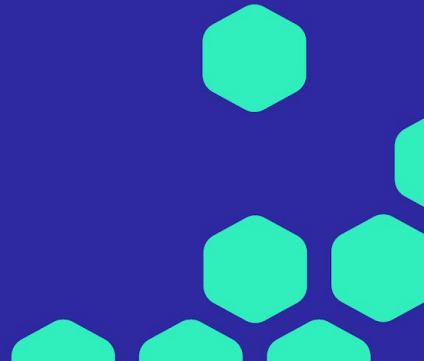
Interfície Collection

La seva definició, extreta de la documentació de Java, és força autoexplicativa atesos els noms que utilitzen:

```
public interface Collection<E> extends Iterable<E> {  
    // Basic operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(E element); //optional  
    boolean remove(Object element); //optional  
    Iterator<E> iterator();  
    // Bulk operations  
    boolean containsAll(Collection<?> c);  
    boolean addAll(Collection<? extends E> c); //optional  
    boolean removeAll(Collection<?> c); //optional  
    boolean retainAll(Collection<?> c); //optional  
    void clear(); //optional  
    // Array operations  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
}
```

Interfície Collection

El llenguatge Java no proporciona cap classe que implementi directament aquesta interfície, sinó que implementa interfícies derivades d'aquesta. Això no és un impediment per implementar directament, quan convingui, aquesta interfície, però la majoria de vegades n'hi ha prou d'implementar les interfícies derivades o, fins i tot, derivar directament de les implementacions que Java proporciona de les diverses interfícies.

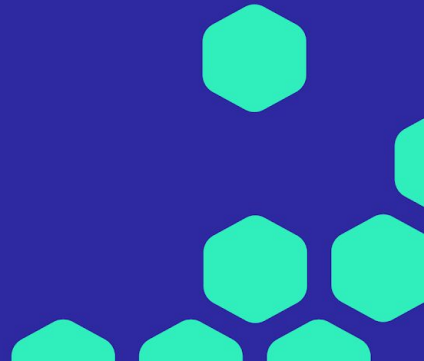


Iterator i ListIterator

Les interfícies de suport Iterator i ListIterator utilitzades per diversos mètodes de les classes que implementen la interfície Collection o les seves subinterfícies permeten recórrer els elements d'una col·lecció.

La definició de la interfície Iterator és:

```
public interface Iterator<E>{  
  
    boolean hasNext() ;  
  
    Object next() ;  
  
    void remove() ;// optional  
  
}
```



Iterator i ListIterator

Així, per exemple, la referència que retorna el mètode `iterator()` de la interfície `Collection` permet recórrer la collecció amb els mètodes `next()` i `hasNext()`, i també permet eliminar l'element actual amb el mètode `remove()` sempre que la collecció permeti l'eliminació dels seus elements.

El comentari optional acompanyant un mètode d'una interfície indica que el mètode pot no estar disponible en alguna de les implementacions de la interfície. Això pot passar si un mètode de la interfície no té sentit en una implementació concreta.

La implementació ha de definir el mètode, però en ser cridat provoca una excepció `UnsupportedOperationException`.



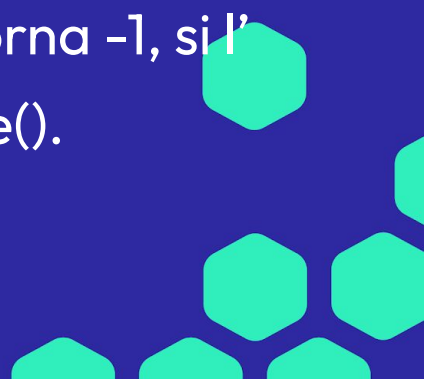
Iterator i ListIterator

La interfície ListIterator permet recórrer els objectes que implementen la interfície List (subinterfície de Collection) en ambdues direccions (endavant i endarrere) i efectuar algunes modificacions mentre s'efectua el recorregut. Els mètodes que defineix la interfície són:

```
public interface ListIterator<E> extends Iterator<E>{  
    boolean hasNext();  
    E next();  
    boolean hasPrevious();  
    E previous();  
    int nextIndex();  
    int previousIndex();  
    void remove();  
    void set(E e);  
    void add(E e);  
  
}
```


Iterator i ListIterator

El recorregut pels elements de la col·lecció (llista) s'efectua amb els mètodes `next()` i `previous()`. En una llista amb n elements, els elements es numeren de 0 a $n-1$, però els valors vàlids per a l'índex iterador són de 0 a n , de manera que l'índex x es troba entre els elements $x-1$ i x , per tant, el mètode `previousIndex()` retorna $x-1$ i el mètode `nextIndex()` retorna x ; si l'índex és 0, `previousIndex()` retorna -1 , si l'índex és n , `nextIndex()` retorna el resultat del mètode `size()`.



Interfícies Comparable i Comparator

Ordering objects

```
public interface Comparable<T> {  
    int compareTo(T other);  
}
```

Instances of classes that implement this interface are *comparable to each other*.

```
public interface Comparator<T> {  
    int compare(T lhs, T rhs);  
    boolean equals(Object other);  
}
```

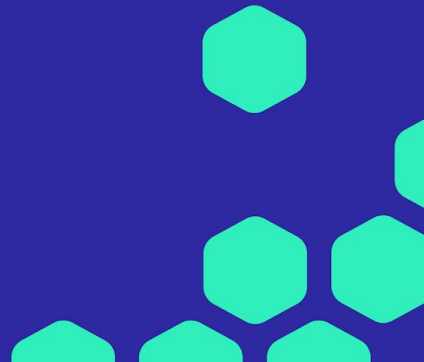
Instances of classes that implement this interface can *compare objects of type T*.

```
return  
<   -1  
==   0  
>   1
```

```
class Dog implements Comparable{  
    private int size;  
    Dog(int s){  
        size = s;  
    }  
  
    public int getSize(){  
        return size;  
    }  
  
    public int compareTo(Object o) {  
        Dog d = (Dog)o;  
        int r = 0;  
        if(size < d.getSize()) r = -1;  
        if(size == d.getSize()) r = 0;  
        if(size > d.getSize()) r = 1;  
        return r;  
    }  
}
```

Interfícies Comparable i Comparator

Les interfícies de suport Comparable i Comparator estan orientades a mantenir una relació d'ordre en les classes del framework de colleccions que implementen interfícies que faciliten ordenació (List, SortedSet i SortedMap).



Interfícies Comparable i Comparator

La interfície Comparable consta d'un únic mètode:

```
public interface Comparable<T> {  
    int compareTo(T o);  
}
```

El mètode compareTo() compara l'objecte sobre el qual s'aplica el mètode amb l'objecte o rebut per paràmetre i retorna un enter negatiu, zero o positiu en funció de si l'objecte sobre el qual s'aplica el mètode és menor, igual o major que l'objecte rebut per paràmetre. Si els dos objectes no són comparables, el mètode ha de generar una excepció ClassCastException.



Interfícies Comparable i Comparator

El llenguatge Java proporciona un munt de classes que implementen aquesta interfície (String, Character, Date, Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger...).

Per tant, totes aquestes classes proporcionen una implementació del mètode `compareTo()`.

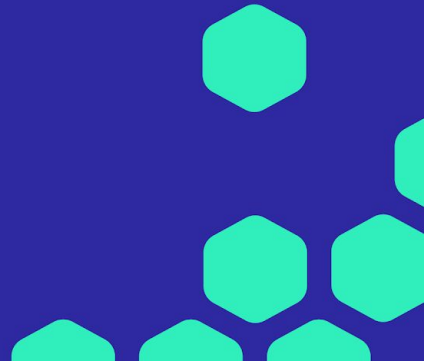
Tota implementació del mètode `compareTo()` hauria de ser compatible amb el mètode `equals()`, de manera que `compareTo()` retorni zero únicament si `equals()` retorna true i, a més, hauria de verificar la propietat transitiva, com en qualsevol relació d'ordre.



Interfícies Comparable i Comparator

De les classes que implementen la interfície Comparable es diu que tenen un ordre natural.

Les colleccions de classes que implementen la interfície Comparable es poden ordenar amb els mètodes static `Collections.sort()` i `Arrays.sort()`.



Interfícies Comparable i Comparator

Comparator permet ordenar conjunts d'objectes que pertanyen a classes diferents. Per establir un ordre en aquests casos, el programador ha de subministrar un objecte d'una classe que implementi aquesta interfície:

```
public interface Comparator<T> {  
    int compare(T o1, T o2);  
    boolean equals(Object obj);  
}
```

L'objectiu del mètode equals() és comparar objectes Comparator. En canvi, l'objectiu del mètode compare() és comparar dos objectes de diferents classes i obtenir un enter negatiu, zero o positiu, en funció de si l'objecte rebut per primer paràmetre és menor, igual o major que l'objecte rebut per segon paràmetre.

Exemple amb Fruites

<https://www.mkymong.com/java/java-object-sorting-example-comparable-and-comparator/>

Interfícies Set i SortedSet

La interfície `Set<E>` és destinada a col·leccions que no mantenen cap ordre d'inserció i que no poden tenir dos o més objectes iguals. Correspon al concepte matemàtic de conjunt.

El conjunt de mètodes que defineix aquesta interfície és idèntic al conjunt de mètodes definits per la interfície `Collection<E>`.

Les implementacions de la interfície `Set<E>` necessiten el mètode `equals()` per veure si dos objectes del tipus de la col·lecció són iguals i, per tant, no permetre'n la coexistència dins la col·lecció.

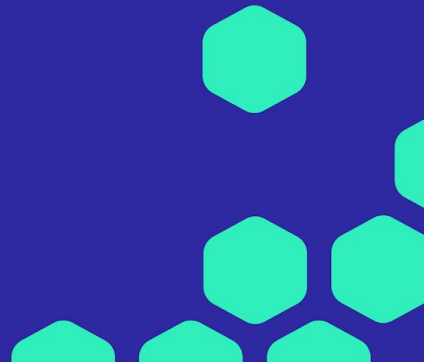


Interfícies Set i SortedSet

La crida `set.equals(Object obj)` retorna `true` si `obj` també és una instància que implementi la interfície `Set`, els dos objectes (`set` i `obj`) tenen el mateix nombre d'elements i tots els elements d'`obj` estan continguts en `set`. Per tant, el resultat pot ser `true` malgrat que `set` i `obj` siguin de classes diferents però que implementen la interfície `Set`.

Els mètodes de la interfície `Set` permeten realitzar les operacions algebraïques unió, intersecció i diferència:

- `s1.containsAll(s2)` permet saber si `s2` està contingut en `s1`.
- `s1.addAll(s2)` permet convertir `s1` en la unió d'`s1` i `s2`.
- `s1.retainAll(s2)` permet convertir `s1` en la intersecció d'`s1` i `s2`.
- `s1.removeAll(s2)` permet convertir `s1` en la diferència d'`s1` i `s2`.



Interfícies Set i SortedSet

La interfície SortedSet derivada de Set afegeix mètodes per permetre gestionar conjunts ordenats. La seva definició és:

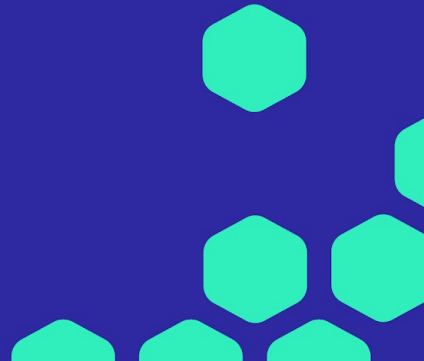
```
public interface SortedSet<E> extends Set<E> {  
    // Range-view  
    SortedSet<E> subSet(E fromElement, E toElement);  
    SortedSet<E> headSet(E toElement);  
    SortedSet<E> tailSet(E fromElement);  
    // Endpoints  
    E first();  
    E last();  
    // Comparator access  
    Comparator<? super E> comparator();  
}
```

Interfícies Set i SortedSet

La relació d'ordre a aplicar sobre els elements d'un objecte col·lecció que implementi la interfície SortedSet es defineix en el moment de la seva construcció, indicant una referència a un objecte que implementi la interfície Comparator. En cas de no indicar cap referència, els elements de l'objecte es comparen amb l'ordre natural.

El mètode comparator() retorna una referència a l'objecte Comparator que defineix l'ordre dels elements del mètode, i retorna null si es tracta de l'ordre natural.

En un objecte SortedSet, els mètodes iterator(), toArray() i toString() gestionen els elements segons l'ordre establert en l'objecte.



Interfície List

La interfície `List<E>` es destina a col·leccions que mantenen l'ordre d'inserció i que poden tenir elements repetits. Per aquest motiu, aquesta interfície declara mètodes addicionals (als definits en la interfície `Collection`) que tenen a veure amb l'ordre i l'accés a elements o interval d'elements:

```
public interface List<E> extends Collection<E> {  
    // Positional access  
    E get(int index);  
    E set(int index, E element); //optional  
    void add(int index, E element); //optional  
    E remove(int index); //optional  
    boolean addAll(int index, Collection<? extends E> c);  
    // Search  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
    // Iteration  
    ListIterator<E> listIterator();  
    ListIterator<E> listIterator(int index);  
    // Range-view  
    List<E> subList(int from, int to);  
}
```

Interfície List

La crida `list.equals(Object obj)` retorna `true` si `obj` també és una instància que implementa la interfície `List`, i els dos objectes tenen el mateix nombre d'elements i contenen elements iguals i en el mateix ordre. Per tant, el resultat pot ser `true` malgrat que `list` i `obj` siguin de classes diferents però que implementen la interfície `List`.

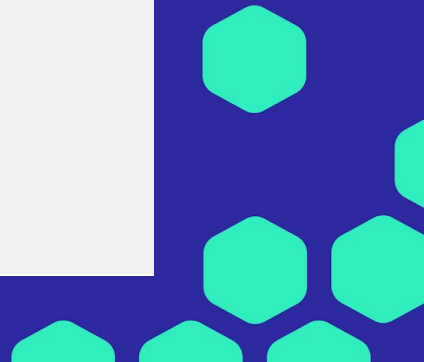
El mètode `add(E o)` definit en la interfície `Collection` afegeix l'element `o` pel final de la llista, i el mètode `remove(Object o)` definit en la interfície `Collection` elimina la primera aparició de l'objecte indicat.



Interfície Queue

La interfície Queue<E> es destina a gestionar col·leccions que guarden múltiples elements abans de ser processats i, per aquest motiu, afegeix els mètodes següents als definits en la interfície Collection:

```
public interface Queue<E> extends Collection<E> {  
    E element();  
    boolean offer(E e);  
    E peek();  
    E poll();  
    E remove();  
}
```



Interfície Queue

En informàtica, quan es parla de cues, es pensa sempre en una gestió dels elements amb un algorisme FIFO (first input, first output –primer en entrar, primer en sortir–), però això no és obligatòriament així en les classes que implementen la interfície Queue. Un exemple són les cues amb prioritat, en què els elements s'ordenen segons un valor per a cada element. Per tant, les implementacions d'aquesta interfície han de definir l'ordre dels seus elements si no es tracta d'implementacions FIFO.

Els mètodes típics en la gestió de cues (encuar, desencuar i inici) prenen, en la interfície Queue, dues formes segons la reacció davant una fallada en l'operació: mètodes que retornen una excepció i mètodes que retornen un valor especial (null o false, segons l'operació). La taula següent ens mostra la classificació dels mètodes segons aquests criteris.

Interfície Queue

Operació	Mètode que provoca excepció	Mètode que retorna un valor especial
Encuar	boolean add (E e)	boolean offer(E e)
Desencuar	E remove()	E poll()
Inici	E element()	E peek()

Interfícies Map i SortedMaps

La interfície Map<E> es destina a gestionar agrupacions d'elements als quals s'accedeix mitjançant una clau, la qual ha de ser única per als diferents elements de l'agrupació. La definició és:

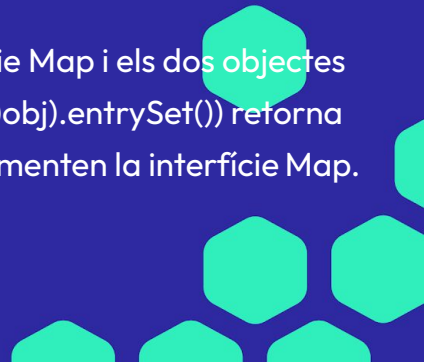
```
public interface Map<K,V> {  
    // Basic operations  
    V put(K key, V value); //optional  
    V get(Object key);  
    V remove(Object key); //optional  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    int size();  
    boolean isEmpty();  
    // Bulk operations  
    void putAll(Map<? extends K, ? extends V> m); // optional  
    void clear(); // optional  
    // Collection Views  
    public Set<K> keySet();  
    public Collection<V> values();  
    public Set<Map.Entry<K,V>> entrySet();  
    // Interface for entrySet elements  
    public interface Entry {  
        K getKey();  
        V getValue();  
        V setValue(V value);  
    }  
}
```

Interfícies Map i SortedMaps

Molts d'aquests mètodes tenen un significat evident, però altres no tant.

El mètode `entrySet()` retorna una visió del Map com a Set. Els elements d'aquest Set són referències a la interfície `Map.Entry` que és una interfície interna de Map que permet modificar i eliminar elements del Map, però no afegir-hi nous elements. El mètode `get(key)` permet obtenir l'element a partir de la clau. El mètode `keySet()` retorna una visió de les claus com a Set. El mètode `values()` retorna una visió dels elements del Map com a Collection (perquè hi pot haver elements repetits i com a Set això no seria factible). El mètode `put()` permet afegir una parella clau/element mentre que `putAll(map)` permet afegir-hi totes les parelles d'un Map passat per paràmetre (les parelles amb clau nova s'afegeixen i, en les parelles amb clau ja existent en el Map, els elements nous substitueixen els elements existents). El mètode `remove(key)` elimina una parella clau/element a partir de la clau.

La crida `map.equals(Object obj)` retorna `true` si `obj` també és una instància que implementa la interfície Map i els dos objectes representen el mateix mapatge o, dit amb altres paraules, si l'expressió `map.entrySet().equals(((Map)obj).entrySet())` retorna `true`. Per tant, el resultat pot ser `true` malgrat que `map` i `obj` siguin de classes diferents però que implementen la interfície Map.



Interfícies Map i SortedMaps

La interfície SortedMap és una interfície Map que permet mantenir ordenades les seves parelles en ordre ascendent segons el valor de la clau, seguint l'ordre natural o la relació d'ordre proporcionada per un objecte que implementi la interfície Comparator proporcionada en el moment de creació de la instància.

La seva definició, a partir de la interfície Map, és similar a la definició de la interfície SortedSet a partir de la interfície Set:

```
public interface SortedMap<K, V> extends Map<K, V> {  
    // Range-view  
    SortedMap<K, V> subMap(K fromKey, K toKey);  
    SortedMap<K, V> headMap(K toKey);  
    SortedMap<K, V> tailMap(K fromKey);  
    // Endpoints  
    K firstKey();  
    K lastKey();  
    // Comparator access  
    Comparator<? super K> comparator();  
}
```

