java.util

Interface Set

All Superinterfaces:
Collection

All Known Subinterfaces:
SortedSet

All Known Implementing Classes:
AbstractSet, HashSet, LinkedHashSet, TreeSet

public interface **Set**
extends Collection

A collection that contains no duplicate elements. More formally, sets contain no pair of elements `e1` and `e2` such that `e1.equals(e2)`, and at most one null element. As implied by its name, this interface models the mathematical *set* abstraction.

The `Set` interface places additional stipulations, beyond those inherited from the `Collection` interface, on the contracts of all constructors and on the contracts of the `add`, `equals` and `hashCode` methods. Declarations for other inherited methods are also included here for convenience. (The specifications accompanying these declarations have been tailored to the `Set` interface, but they do not contain any additional stipulations.)

The additional stipulation on constructors is, not surprisingly, that all constructors must create a set that contains no duplicate elements (as defined above).

Note: Great care must be exercised if mutable objects are used as set elements. The behavior of a set is not specified if the value of an object is changed in a manner that affects equals comparisons while the object is an element in the set. A special case of this prohibition is that it is not permissible for a set to contain itself as an element.

Some set implementations have restrictions on the elements that they may contain. For example, some implementations prohibit null elements, and some have restrictions on the types of their elements. Attempting to add an ineligible element throws an unchecked exception, typically `NullPointerException` or `ClassCastException`. Attempting to query the presence of an ineligible element may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible element whose completion would not result in the insertion of an ineligible element into the set may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

This interface is a member of the Java Collections Framework.

Since:
1.2

See Also:
`Collection`, `List`, `SortedSet`, `HashSet`, `TreeSet`, `AbstractSet`, `Collections.singleton(java.lang.Object)`, `Collections.EMPTY_SET`

Method Summary

`boolean`	**`add`**`(Object o)` Adds the specified element to this set if it is not already present (optional operation).
`boolean`	**`addAll`**`(Collection c)` Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
`void`	**`clear`**`()` Removes all of the elements from this set (optional operation).
`boolean`	**`contains`**`(Object o)` Returns `true` if this set contains the specified element.
`boolean`	**`containsAll`**`(Collection c)` Returns `true` if this set contains all of the elements of the specified collection.
`boolean`	**`equals`**`(Object o)` Compares the specified object with this set for equality.
`int`	**`hashCode`**`()` Returns the hash code value for this set.
`boolean`	**`isEmpty`**`()` Returns `true` if this set contains no elements.
`Iterator`	**`iterator`**`()` Returns an iterator over the elements in this set.
`boolean`	**`remove`**`(Object o)` Removes the specified element from this set if it is present (optional operation).
`boolean`	**`removeAll`**`(Collection c)` Removes from this set all of its elements that are contained in the specified collection (optional operation).
`boolean`	**`retainAll`**`(Collection c)` Retains only the elements in this set that are contained in the specified collection (optional operation).

`int`	`size()` Returns the number of elements in this set (its cardinality).
`Object[]`	`toArray()` Returns an array containing all of the elements in this set.
`Object[]`	`toArray(Object[] 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.

Method Detail

size

```
public int size()
```

Returns the number of elements in this set (its cardinality). If this set contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

Specified by:
`size` in interface `Collection`

Returns:
the number of elements in this set (its cardinality).

---

isEmpty

```
public boolean isEmpty()
```

Returns `true` if this set contains no elements.

Specified by:
`isEmpty` in interface `Collection`

Returns:
`true` if this set contains no elements.

---

contains

```
public boolean contains(Object o)
```

Returns `true` if this set contains the specified element. More formally, returns `true` if and only if this set contains an element `e` such that `(o==null ? e==null : o.equals(e))`.

Specified by:

`contains` in interface `Collection`

Parameters:

`o` - element whose presence in this set is to be tested.

Returns:

`true` if this set contains the specified element.

Throws:

`ClassCastException` - if the type of the specified element is incompatible with this set (optional).

`NullPointerException` - if the specified element is null and this set does not support null elements (optional).

---

iterator

```
public Iterator iterator()
```

Returns an iterator over the elements in this set. The elements are returned in no particular order (unless this set is an instance of some class that provides a guarantee).

Specified by:

`iterator` in interface `Collection`

Returns:

an iterator over the elements in this set.

---

toArray

```
public Object[] toArray()
```

Returns an array containing all of the elements in this set. Obeys the general contract of the `Collection.toArray` method.

Specified by:

`toArray` in interface `Collection`

Returns:

an array containing all of the elements in this set.

---

toArray

```
public Object[] toArray(Object[] 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. Obeys the general contract of the

`Collection.toArray(Object[])` method.

Specified by:

`toArray` in interface `Collection`

Parameters:

`a` - the array into which the elements of this set are to be stored, if it is big enough; otherwise, a new array of the same runtime type is allocated for this purpose.

Returns:

an array containing the elements of this set.

Throws:

`ArrayStoreException` - the runtime type of a is not a supertype of the runtime type of every element in this set.
`NullPointerException` - if the specified array is `null`.

---

add

`public boolean **add**(Object o)`

Adds the specified element to this set if it is not already present (optional operation). More formally, adds the specified element, `o`, to this set if this set contains no element `e` such that `(o==null ? e==null : o.equals(e))`. If this set already contains the specified element, the call leaves this set unchanged and returns `false`. In combination with the restriction on constructors, this ensures that sets never contain duplicate elements.

The stipulation above does not imply that sets must accept all elements; sets may refuse to add any particular element, including `null`, and throwing an exception, as described in the specification for `Collection.add`. Individual set implementations should clearly document any restrictions on the the elements that they may contain.

Specified by:

`add` in interface `Collection`

Parameters:

`o` - element to be added to this set.

Returns:

`true` if this set did not already contain the specified element.

Throws:

`UnsupportedOperationException` - if the `add` method is not supported by this set.
`ClassCastException` - if the class of the specified element prevents it from being added to this set.
`NullPointerException` - if the specified element is null and this set does not support null elements.
`IllegalArgumentException` - if some aspect of the specified element prevents it from being added to this set.

---

remove

`public boolean remove(Object o)`

Removes the specified element from this set if it is present (optional operation). More formally, removes an element `e` such that `(o==null ? e==null : o.equals(e))`, if the set contains such an element. Returns `true` if the set contained the specified element (or equivalently, if the set changed as a result of the call). (The set will not contain the specified element once the call returns.)

Specified by:

`remove` in interface `Collection`

Parameters:

`o` - object to be removed from this set, if present.

Returns:

true if the set contained the specified element.

Throws:

`ClassCastException` - if the type of the specified element is incompatible with this set (optional).

`NullPointerException` - if the specified element is null and this set does not support null elements (optional).

`UnsupportedOperationException` - if the `remove` method is not supported by this set.

---

containsAll

`public boolean containsAll(Collection c)`

Returns `true` if this set contains all of the elements of the specified collection. If the specified collection is also a set, this method returns `true` if it is a *subset* of this set.

Specified by:

`containsAll` in interface `Collection`

Parameters:

`c` - collection to be checked for containment in this set.

Returns:

`true` if this set contains all of the elements of the specified collection.

Throws:

`ClassCastException` - if the types of one or more elements in the specified collection are incompatible with this set (optional).

`NullPointerException` - if the specified collection contains one or more null elements and this set does not support null elements (optional).

`NullPointerException` - if the specified collection is `null`.

See Also:

`contains(Object)`

addAll

```
public boolean addAll(Collection c)
```

Adds all of the elements in the specified collection to this set if they're not already present (optional operation). If the specified collection is also a set, the `addAll` operation effectively modifies this set so that its value is the *union* of the two sets. The behavior of this operation is unspecified if the specified collection is modified while the operation is in progress.

Specified by:
`addAll` in interface `Collection`

Parameters:
`c` - collection whose elements are to be added to this set.

Returns:
`true` if this set changed as a result of the call.

Throws:
`UnsupportedOperationException` - if the `addAll` method is not supported by this set.
`ClassCastException` - if the class of some element of the specified collection prevents it from being added to this set.
`NullPointerException` - if the specified collection contains one or more null elements and this set does not support null elements, or if the specified collection is `null`.
`IllegalArgumentException` - if some aspect of some element of the specified collection prevents it from being added to this set.

See Also:
`add(Object)`

retainAll

```
public boolean retainAll(Collection c)
```

Retains only the elements in this set that are contained in the specified collection (optional operation). In other words, removes from this set all of its elements that are not contained in the specified collection. If the specified collection is also a set, this operation effectively modifies this set so that its value is the *intersection* of the two sets.

Specified by:
`retainAll` in interface `Collection`

Parameters:
`c` - collection that defines which elements this set will retain.

Returns:
`true` if this collection changed as a result of the call.

Throws:

`UnsupportedOperationException` - if the `retainAll` method is not supported by this Collection.
`ClassCastException` - if the types of one or more elements in this set are incompatible with the specified collection (optional).
`NullPointerException` - if this set contains a null element and the specified collection does not support null elements (optional).
`NullPointerException` - if the specified collection is `null`.

See Also:

`remove(Object)`

---

removeAll

```
public boolean removeAll(Collection c)
```

Removes from this set all of its elements that are contained in the specified collection (optional operation). If the specified collection is also a set, this operation effectively modifies this set so that its value is the *asymmetric set difference* of the two sets.

Specified by:

`removeAll` in interface `Collection`

Parameters:

`c` - collection that defines which elements will be removed from this set.

Returns:

`true` if this set changed as a result of the call.

Throws:

`UnsupportedOperationException` - if the `removeAll` method is not supported by this Collection.
`ClassCastException` - if the types of one or more elements in this set are incompatible with the specified collection (optional).
`NullPointerException` - if this set contains a null element and the specified collection does not support null elements (optional).
`NullPointerException` - if the specified collection is `null`.

See Also:

`remove(Object)`

---

clear

```
public void clear()
```

Removes all of the elements from this set (optional operation). This set will be empty after this call returns (unless it throws an exception).

Specified by:

`clear` in interface `Collection`

Throws:

UnsupportedOperationException - if the `clear` method is not supported by this set.

---

equals

`public boolean **equals**(Object o)`

Compares the specified object with this set for equality. Returns `true` if the specified object is also a set, the two sets have the same size, and every member of the specified set is contained in this set (or equivalently, every member of this set is contained in the specified set). This definition ensures that the equals method works properly across different implementations of the set interface.

Specified by:

equals in interface Collection

Overrides:

equals in class Object

Parameters:

`o` - Object to be compared for equality with this set.

Returns:

`true` if the specified Object is equal to this set.

See Also:

Object.hashCode(), Hashtable

---

hashCode

`public int **hashCode**()`

Returns the hash code value for this set. The hash code of a set is defined to be the sum of the hash codes of the elements in the set, where the hashcode of a `null` element is defined to be zero. This ensures that `s1.equals(s2)` implies that `s1.hashCode()==s2.hashCode()` for any two sets `s1` and `s2`, as required by the general contract of the `Object.hashCode` method.

Specified by:

hashCode in interface Collection

Overrides:

hashCode in class Object

Returns:

the hash code value for this set.

See Also:

Object.hashCode(), Object.equals(Object), equals(Object)
