| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Comparator.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/Collections.html)   [**NEXT CLASS**](http://docs.google.com/java/util/ConcurrentModificationException.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Comparator.html)    [**NO FRAMES**](http://docs.google.com/Comparator.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

## **java.util**

Interface Comparator<T>

**Type Parameters:**T - the type of objects that may be compared by this comparator **All Known Implementing Classes:** [Collator](http://docs.google.com/java/text/Collator.html), [RuleBasedCollator](http://docs.google.com/java/text/RuleBasedCollator.html)

public interface **Comparator<T>**

A comparison function, which imposes a *total ordering* on some collection of objects. Comparators can be passed to a sort method (such as [Collections.sort](http://docs.google.com/java/util/Collections.html#sort(java.util.List,%20java.util.Comparator)) or [Arrays.sort](http://docs.google.com/java/util/Arrays.html#sort(T%5B%5D,%20java.util.Comparator))) to allow precise control over the sort order. Comparators can also be used to control the order of certain data structures (such as [sorted sets](http://docs.google.com/java/util/SortedSet.html) or [sorted maps](http://docs.google.com/java/util/SortedMap.html)), or to provide an ordering for collections of objects that don't have a [natural ordering](http://docs.google.com/java/lang/Comparable.html).

The ordering imposed by a comparator c on a set of elements S is said to be *consistent with equals* if and only if c.compare(e1, e2)==0 has the same boolean value as e1.equals(e2) for every e1 and e2 in S.

Caution should be exercised when using a comparator capable of imposing an ordering inconsistent with equals to order a sorted set (or sorted map). Suppose a sorted set (or sorted map) with an explicit comparator c is used with elements (or keys) drawn from a set S. If the ordering imposed by c on S is inconsistent with equals, the sorted set (or sorted map) will behave "strangely." In particular the sorted set (or sorted map) will violate the general contract for set (or map), which is defined in terms of equals.

For example, suppose one adds two elements a and b such that (a.equals(b) && c.compare(a, b) != 0) to an empty TreeSet with comparator c. The second add operation will return true (and the size of the tree set will increase) because a and b are not equivalent from the tree set's perspective, even though this is contrary to the specification of the [Set.add](http://docs.google.com/java/util/Set.html#add(E)) method.

Note: It is generally a good idea for comparators to also implement java.io.Serializable, as they may be used as ordering methods in serializable data structures (like [TreeSet](http://docs.google.com/java/util/TreeSet.html), [TreeMap](http://docs.google.com/java/util/TreeMap.html)). In order for the data structure to serialize successfully, the comparator (if provided) must implement Serializable.

For the mathematically inclined, the *relation* that defines the *imposed ordering* that a given comparator c imposes on a given set of objects S is:

{(x, y) such that c.compare(x, y) <= 0}.

The *quotient* for this total order is:

{(x, y) such that c.compare(x, y) == 0}.

It follows immediately from the contract for compare that the quotient is an *equivalence relation* on S, and that the imposed ordering is a *total order* on S. When we say that the ordering imposed by c on S is *consistent with equals*, we mean that the quotient for the ordering is the equivalence relation defined by the objects' [equals(Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) method(s):

{(x, y) such that x.equals(y)}.

This interface is a member of the  [Java Collections Framework](http://docs.google.com/technotes/guides/collections/index.html).

**Since:** 1.2 **See Also:**[Comparable](http://docs.google.com/java/lang/Comparable.html), [Serializable](http://docs.google.com/java/io/Serializable.html)

| **Method Summary** | |
| --- | --- |
| int | [**compare**](http://docs.google.com/java/util/Comparator.html#compare(T,%20T))([T](http://docs.google.com/java/util/Comparator.html) o1, [T](http://docs.google.com/java/util/Comparator.html) o2)            Compares its two arguments for order. |
| boolean | [**equals**](http://docs.google.com/java/util/Comparator.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Indicates whether some other object is "equal to" this comparator. |

| **Method Detail** |
| --- |

### compare

int **compare**([T](http://docs.google.com/java/util/Comparator.html) o1,  
 [T](http://docs.google.com/java/util/Comparator.html) o2)

Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

In the foregoing description, the notation sgn(*expression*) designates the mathematical *signum* function, which is defined to return one of -1, 0, or 1 according to whether the value of *expression* is negative, zero or positive.

The implementor must ensure that sgn(compare(x, y)) == -sgn(compare(y, x)) for all x and y. (This implies that compare(x, y) must throw an exception if and only if compare(y, x) throws an exception.)

The implementor must also ensure that the relation is transitive: ((compare(x, y)>0) && (compare(y, z)>0)) implies compare(x, z)>0.

Finally, the implementor must ensure that compare(x, y)==0 implies that sgn(compare(x, z))==sgn(compare(y, z)) for all z.

It is generally the case, but *not* strictly required that (compare(x, y)==0) == (x.equals(y)). Generally speaking, any comparator that violates this condition should clearly indicate this fact. The recommended language is "Note: this comparator imposes orderings that are inconsistent with equals."

**Parameters:**o1 - the first object to be compared.o2 - the second object to be compared. **Returns:**a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second. **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the arguments' types prevent them from being compared by this comparator.

### equals

boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) obj)

Indicates whether some other object is "equal to" this comparator. This method must obey the general contract of [Object.equals(Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)). Additionally, this method can return true *only* if the specified object is also a comparator and it imposes the same ordering as this comparator. Thus, comp1.equals(comp2) implies that sgn(comp1.compare(o1, o2))==sgn(comp2.compare(o1, o2)) for every object reference o1 and o2.

Note that it is *always* safe *not* to override Object.equals(Object). However, overriding this method may, in some cases, improve performance by allowing programs to determine that two distinct comparators impose the same order.

**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**obj - the reference object with which to compare. **Returns:**true only if the specified object is also a comparator and it imposes the same ordering as this comparator.**See Also:**[Object.equals(Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode())

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Comparator.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/Collections.html)   [**NEXT CLASS**](http://docs.google.com/java/util/ConcurrentModificationException.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Comparator.html)    [**NO FRAMES**](http://docs.google.com/Comparator.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).