GUAVA - ORDERING CLASS

Ordering can be seen as an enriched comparator with enhanced chaining functionality, multiple utility methods, multi-type sorting capability, etc.

Class Declaration

Following is the declaration for **com.google.common.collect.Ordering<T>** class:

```
@GwtCompatible
public abstract class Ordering<T>
   extends Object
      implements Comparator<T>
```

Class Methods

Sr.No	Method & Description
1	**static Ordering<Object> allEqual** Returns an ordering which treats all values as equal, indicating "no ordering." Passing this ordering to any stable sort algorithm results in no change to the order of elements.
2	**static Ordering<Object> arbitrary** Returns an arbitrary ordering over all objects, for which comparea, b == 0 implies a == b $identityequality$.
3	**int binarySearch***List < ? extendsT > sortedList, Tkey* Searches sortedList for key using the binary search algorithm.
4	**abstract int compare***Tleft, Tright* Compares its two arguments for order.
5	**<U extends T> Ordering<U> compound***Comparator < ? superU > secondaryComparator* Returns an ordering which first uses the ordering this, but which in the event of a "tie", then delegates to secondaryComparator.
6	**static <T> Ordering<T> compound** ***Iterable*** **< ? extendsComparator < ? superT >> comparators** Returns an ordering which tries each given comparator in order until a non-zero result is found, returning that result, and returning zero only if all comparators return zero.
7	**static <T> Ordering<T> explicit***List < T > valuesInOrder* Returns an ordering that compares objects according to the order in which they appear in the given list.
8	**static <T> Ordering<T> explicit***TleastValue, T. . . remainingValuesInOrder*

Returns an ordering that compares objects according to the order in which they are given to this method.

9 **static <T> Ordering<T> from***Comparator < T > comparator*

Returns an ordering based on an existing comparator instance.

10 **<E extends T> List<E> greatestOf***Iterable < E > iterable, intk*

Returns the k greatest elements of the given iterable according to this ordering, in order from greatest to least.

11 **<E extends T> List<E> greatestOf***Iterator < E > iterator, intk*

Returns the k greatest elements from the given iterator according to this ordering, in order from greatest to least.

12 **<E extends T> ImmutableList<E> immutableSortedCopy***Iterable < E > elements*

Returns an immutable list containing elements sorted by this ordering.

13 **boolean isOrdered***Iterable < ?extendsT > iterable*

Returns true if each element in iterable after the first is greater than or equal to the element that preceded it, according to this ordering.

14 **boolean isStrictlyOrdered***Iterable < ?extendsT > iterable*

Returns true if each element in iterable after the first is strictly greater than the element that preceded it, according to this ordering

15 **<E extends T> List<E> leastOf***Iterable < E > iterable, intk*

Returns the k least elements of the given iterable according to this ordering, in order from least to greatest.

16 **<E extends T> List<E> leastOf***Iterator < E > elements, intk*

Returns the k least elements from the given iterator according to this ordering, in order from least to greatest.

17 **<S extends T> Ordering<Iterable<S>> lexicographical**

Returns a new ordering which sorts iterables by comparing corresponding elements pairwise until a nonzero result is found; imposes "dictionary order".

18 **<E extends T> E max***Ea, Eb*

Returns the greater of the two values according to this ordering.

19 **<E extends T> E max***Ea, Eb, Ec, E. . . rest*

Returns the greatest of the specified values according to this ordering.

20 **<E extends T> E max***Iterable < E > iterable*

Returns the greatest of the specified values according to this ordering.

21 **<E extends T> E max***Iterator < E > iterator*

Returns the greatest of the specified values according to this ordering.

22 **<E extends T> E min***Ea, Eb*

Returns the lesser of the two values according to this ordering.

23 **<E extends T> E min***Ea, Eb, Ec, E. . . rest*

Returns the least of the specified values according to this ordering.

24 **<E extends T> E min***Iterable < E > iterable*

Returns the least of the specified values according to this ordering.

25 **<E extends T> E min***Iterator < E > iterator*

Returns the least of the specified values according to this ordering.

26 **static <C extends Comparable> Ordering<C> natural**

Returns a serializable ordering that uses the natural order of the values.

27 **<S extends T> Ordering<S> nullsFirst**

Returns an ordering that treats null as less than all other values and uses this to compare non-null values.

28 **<S extends T> Ordering<S> nullsLast**

Returns an ordering that treats null as greater than all other values and uses this ordering to compare non-null values.

29 **<F> Ordering<F> onResultOf***Function < F, ? extendsT > function*

Returns a new ordering on F which orders elements by first applying a function to them, then comparing those results using this.

30 **<S extends T> Ordering<S> reverse**

Returns the reverse of this ordering; the Ordering equivalent to Collections.reverseOrder*Comparator*.

31 **<E extends T> List<E> sortedCopy***Iterable < E > elements*

Returns a mutable list containing elements sorted by this ordering; use this only when the resulting list may need further modification, or may contain null.

32 **static Ordering<Object> usingToString**

Returns an ordering that compares objects by the natural ordering of their string representations as returned by toString.

Methods Inherited

This class inherits methods from the following class:

- java.lang.Object

Example of Ordering class

Create the following java program using any editor of your choice in say **C:/> Guava.**

GuavaTester.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import com.google.common.collect.Ordering;

public class GuavaTester {
   public static void main(String args[]) {
      List<Integer> numbers = new ArrayList<Integer>();

      numbers.add(new Integer(5));
      numbers.add(new Integer(2));
      numbers.add(new Integer(15));
      numbers.add(new Integer(51));
      numbers.add(new Integer(53));
      numbers.add(new Integer(35));
      numbers.add(new Integer(45));
      numbers.add(new Integer(32));
      numbers.add(new Integer(43));
      numbers.add(new Integer(16));

      Ordering ordering = Ordering.natural();
      System.out.println("Input List: ");
      System.out.println(numbers);

      Collections.sort(numbers,ordering );
      System.out.println("Sorted List: ");
      System.out.println(numbers);

      System.out.println("======================");
      System.out.println("List is sorted: " + ordering.isOrdered(numbers));
      System.out.println("Minimum: " + ordering.min(numbers));
      System.out.println("Maximum: " + ordering.max(numbers));

      Collections.sort(numbers,ordering.reverse());
      System.out.println("Reverse: " + numbers);

      numbers.add(null);
      System.out.println("Null added to Sorted List: ");
      System.out.println(numbers);

      Collections.sort(numbers,ordering.nullsFirst());
      System.out.println("Null first Sorted List: ");
      System.out.println(numbers);
      System.out.println("======================");

      List<String> names = new ArrayList<String>();

      names.add("Ram");
      names.add("Shyam");
      names.add("Mohan");
      names.add("Sohan");
      names.add("Ramesh");
      names.add("Suresh");
      names.add("Naresh");
      names.add("Mahesh");
      names.add(null);
      names.add("Vikas");
      names.add("Deepak");
```

```
      System.out.println("Another List: ");
      System.out.println(names);

      Collections.sort(names,ordering.nullsFirst().reverse());
      System.out.println("Null first then reverse sorted list: ");
      System.out.println(names);
   }
}
```

Verify the Result

Compile the class using **javac** compiler as follows:

```
C:\Guava>javac GuavaTester.java
```

Now run the GuavaTester to see the result.

```
C:\Guava>java GuavaTester
```

See the result.

```
Input List:
[5, 2, 15, 51, 53, 35, 45, 32, 43, 16]
Sorted List:
[2, 5, 15, 16, 32, 35, 43, 45, 51, 53]
======================
List is sorted: true
Minimum: 2
Maximum: 53
Reverse: [53, 51, 45, 43, 35, 32, 16, 15, 5, 2]
Null added to Sorted List:
[53, 51, 45, 43, 35, 32, 16, 15, 5, 2, null]
Null first Sorted List:
[null, 2, 5, 15, 16, 32, 35, 43, 45, 51, 53]
======================
Another List:
[Ram, Shyam, Mohan, Sohan, Ramesh, Suresh, Naresh, Mahesh, null, Vikas, Deepak]
Null first then reverse sorted list:
[Vikas, Suresh, Sohan, Shyam, Ramesh, Ram, Naresh, Mohan, Mahesh, Deepak, null]
```