Collections max method

max(..) method is overloaded method in Collections class. Previously we had studied methods like [swap(..)](http://data-structure-learning.blogspot.com/2015/06/collections-class-swap-method.html), [synchronized](http://data-structure-learning.blogspot.com/2015/06/synchronized-collections-by-collections.html), [binarySearch(..)](http://data-structure-learning.blogspot.com/2015/06/collections-class-binarysearch-method.html), [copy(..)](http://data-structure-learning.blogspot.com/2015/06/collections-class-copy-method.html), [fill(..)](http://data-structure-learning.blogspot.com/2015/06/collections-class-fill-method.html), [addAll(..)](http://data-structure-learning.blogspot.com/2015/05/collections-class-addall-method.html), [min(..) with Comparable](http://data-structure-learning.blogspot.com/2015/06/collections-min-method-with-comparable.html) and [min(..) with Comparator](http://data-structure-learning.blogspot.com/2015/06/collections-min-method-with-comparator.html).

To understand this method you need to know the [Comparable](http://data-structure-learning.blogspot.com/2015/06/comparable-interface-for-sorting.html) and [Comparator](http://data-structure-learning.blogspot.com/2015/06/comparator-interface.html) Interfaces and [differences between them](http://data-structure-learning.blogspot.com/2015/06/difference-between-comparable-and.html). I would highly recommend that you read those posts.

There are two versions of max(..) method. In this post we will cover only one of them. In next post we will discuss other one. The difference is that one method takes Comparable<T> interface and another takes Comparator<T> Interface as natural ordering method.

First we will cover the method that expects us to write Comparable<T> interface. All elements must have implemented Comparable<T> interface and they must be comparable i.e. e1.compareTo(e2) must not throw ClassCastException for any elements in collection. We will take 2 examples for this method. One in which class (Integer, String) has Comparable implementation provided and another in which we will implement our own Comparable interface.

Below is the code for max(..) method in Collections class.

**public** **static** <T **extends** Object & Comparable<? **super** T>> T max(

Collection<? **extends** T> coll) {

/\*\*

\* Take iterator of the collection

\* \*/

Iterator<? **extends** T> i = coll.iterator();

//This statement is used for comparison with he candidate.

T candidate = i.next();

**while** (i.hasNext()) {

T next = i.next();

/\*\*

\* next and candidate are compared.

\* if next is small it becomes candidate for comparision

\* for next elements.

\* \*/

**if** (next.compareTo(candidate) > 0)

candidate = next;

}

**return** candidate;

}

If you see the method signature <T **extends** Object & Comparable<? **super** T>> it seems to be confusing. But it is really not. Let us extract bits from it and understand it.

Comparable<? **super** T> - ? super T is wild card used so as to compare super type of T.

T **extends** Object - this means that type T must extend the class Object. This is used because of backward compatibility.

T **extends** Object & Comparable<? **super** T> - combining it together type t must extend class Object and Comparable of super type of T.

For class String and Integer, they have Comparable<T> implemented in it. We do not need to do much for finding max for it.

**public** **static** **void** maxDemo() {

List<String> list = *populateLanguages*();

System.***out***.println(list);

String str = Collections.*max*(list);

System.***out***.println("max is "+str);

List<Integer> ints=**new** ArrayList<Integer>(Arrays.*asList*(6, 5, 4, 3, 3, 2, 5, 8));

**int** max=Collections.*max*(ints);

System.***out***.println(ints);

System.***out***.println("max is "+max);

}

Output:

[Java, JavaScript, C#, Python]

max is Python

[6, 5, 4, 3, 3, 2, 5, 8]

max is 8