Collections class sort method (Comparator for nulls)

In previous two posts we saw [how to use Comparable interface](http://data-structure-learning.blogspot.com/2015/11/collections-class-sort-method.html) in Collections.sort(List<T>) and [how to use Comparator](http://data-structure-learning.blogspot.com/2015/11/collections-class-sortlist-comparator.html) interface in Collections.sort(List<T> list, Comparator<? super T> c).

Both the previous post has one problem in common. If there is/are null elements in List<T> then Collections.sort(..) will throw NullPointerException.

So in this post we will see how to write a Comparator that will skip the null elements and Compare only the actual elements (elements that are not null).

Following is one way to write conditions in a comparator.

1. If first object is null and second object is null then return 0. Because if both are null means both are same.
2. If first object is null and second object is not null return -1. This step assumes that elements that are null are smaller than elements that are not null. (If you return 1 then all null elements will be at the end of List).
3. If first object is not null and second object is null return 1. Because first object contains value and second object is null. This step is reverse of Step 2.
4. If both the objects contains value then return object1.compareTo(object2).

Below is the code for the Collections.sort with Comparator.

**public** **void** sortForNulls(**final** List<String> names) {

Collections.*sort*(names, **new** Comparator<String>() {

@Override

**public** **int** compare(**final** String name1,

**final** String name2) {

**if** (name1 == **null**) {

**if** (name2 == **null**) {

//Step 1

**return** 0;

}

//Step 2

**return** -1;

} **else** **if** (name2 == **null**) {

//Step 3

**return** 1;

}

//Step 4

**return** name1.compareTo(name2);

}

});

}

Watch out the comments that are written. It describes the 4 different scenarios.

Let us write entire code with

**public** **class** ListSortComparator {

**public** **static** **void** main(String[] args) {

ListSortComparator lsc = **new** ListSortComparator();

List<String> names = lsc.names();

lsc.sortForNulls(names);

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

}

**public** List<String> names() {

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

names.add("Sansa");

names.add("Ramsay");

names.add("Eddard");

names.add("Benjen");

names.add("Robb");

names.add("Catelyn");

names.add(**null**);

names.add("Rickon");

names.add(**null**);

names.add("Brandon");

**return** names;

}

**public** **void** sortForNulls(**final** List<String> names) {

Collections.*sort*(names, **new** Comparator<String>() {

@Override

**public** **int** compare(**final** String name1,

**final** String name2) {

**if** (name1 == **null**) {

**if** (name2 == **null**) {

//Step 1

**return** 0;

}

//Step 2

**return** -1;

} **else** **if** (name2 == **null**) {

//Step 3

**return** 1;

}

//Step 4

**return** name1.compareTo(name2);

}

});

}

}

Output:

Before Sort: [Sansa, Ramsay, Eddard, Benjen, Robb, Catelyn, null, Rickon, null, Brandon]

After Sort: [null, null, Benjen, Brandon, Catelyn, Eddard, Ramsay, Rickon, Robb, Sansa]