ArrayList class sort method

This is 22nd post in series of ArrayList class. Previously we have seen [ArrayList introduction](http://data-structure-learning.blogspot.com/2015/08/arraylist-class-introduction-and-how-it.html), ArrayList class [constructors](http://data-structure-learning.blogspot.com/2015/08/arraylist-class-constructors.html), [add](http://data-structure-learning.blogspot.com/2015/09/arraylist-class-add-methods.html)() method, [addAll](http://data-structure-learning.blogspot.com/2015/09/arraylist-class-addall-methods.html)() method, [clear](http://data-structure-learning.blogspot.com/2015/09/arraylist-class-clear-method.html)() method, [indexOf](http://data-structure-learning.blogspot.com/2015/09/arraylist-class-indexof-method.html)() method, [contains](http://data-structure-learning.blogspot.com/2015/09/arraylist-class-contains-method.html)() method, forEach() method, get(), isEmpty(), iterator(), lastIndexOf() method, listIterator(), remove(int index), remove(Object o), removeAll(Collection<?> c), removeIf(Predicate<? super T> E), replaceAll(UnaryOperator<E> operator) method, retainAll(Collection<?> c) set(), and size() method.

To understand this method you need to know [Comparable](http://data-structure-learning.blogspot.com/2015/06/comparable-interface-for-sorting.html)<E>, [Comparator](http://data-structure-learning.blogspot.com/2015/06/comparator-interface.html)<E> interface and [difference between](http://data-structure-learning.blogspot.com/2015/06/difference-between-comparable-and.html) them. I would highly recommend you to read those posts as it is pre-requisite for this post.

If you are reading this line then I assume that you have read the above posts. So sort() method is new as it is written in ArrayList class in Java 8 version.

Previously to this if we need to sort the List then we can use Collections class.

Collections.sort(list);

As of Java 8 we can directly use sort method provided in ArrayList class or sort() default method in List interface. Yes, as of Java 8 we can define a method (method with body) in interface. sort() method accepts Comparator but we can keep it null. We will see how to use it later.

If there are any null values in List then it will throw NullPointerException.

Let us first assume that there are no null values. Below program sorts the List of Strings. And if you are wondering that how did it sorted then you must read [Comparable](http://data-structure-learning.blogspot.com/2015/06/comparable-interface-for-sorting.html)<E>, [Comparator](http://data-structure-learning.blogspot.com/2015/06/comparator-interface.html)<E> interface and [difference between](http://data-structure-learning.blogspot.com/2015/06/difference-between-comparable-and.html) them again. If any class implements Comparable Interface then it can be sorted directly. See the class definition of String class [here](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html).

Below is the code with List<String> and no nulls.

**package** org.example.collections.list.arraylist;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** ArrayListSortDemo {

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

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

kids.add("Robb");

kids.add("Sansa");

kids.add("John");

kids.add("Arya");

**return** kids;

}

**public** **void** sort(List<String> kidsName) {

System.***out***.println("Unsorted List: " + kidsName);

kidsName.sort(**null**);

System.***out***.println("Sorted List: " + kidsName);

}

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

ArrayListSortDemo sortDemo = **new** ArrayListSortDemo();

List<String> kidsName = sortDemo.kidsNames();

sortDemo.sort(kidsName);

}

}

Output

Unsorted List: [Robb, Sansa, John, Arya]

Sorted List: [Arya, John, Robb, Sansa]

Now let us insert null values in list and run it.

**package** org.example.collections.list.arraylist;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** ArrayListSortDemo {

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

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

kids.add("Robb");

kids.add("Sansa");

kids.add("John");

kids.add(**null**);//null inserted

kids.add("Arya");

**return** kids;

}

**public** **void** sort(List<String> kidsName) {

System.***out***.println("Unsorted List: " + kidsName);

kidsName.sort(**null**);//Comparator is null

System.***out***.println("Sorted List: " + kidsName);

}

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

ArrayListSortDemo sortDemo = **new** ArrayListSortDemo();

List<String> kidsName = sortDemo.kidsNames();

sortDemo.sort(kidsName);

}

}

Output

Unsorted List: [Robb, Sansa, John, null, Arya]

Exception in thread "main" java.lang.NullPointerException

at java.util.ComparableTimSort.binarySort(Unknown Source)

at java.util.ComparableTimSort.sort(Unknown Source)

at java.util.Arrays.sort(Unknown Source)

at java.util.Arrays.sort(Unknown Source)

at java.util.ArrayList.sort(Unknown Source)

at org.example.collections.list.arraylist.ArrayListSortDemo.sort(ArrayListSortDemo.java:20)

at org.example.collections.list.arraylist.ArrayListSortDemo.main(ArrayListSortDemo.java:28)

It throws NullPointerException.

To avoid this surprise we can use Comparator.

**public** **void** sort(List<String> kidsName) {

System.***out***.println("Unsorted List: " + kidsName);

kidsName.sort(**new** Comparator<String>() {

@Override

**public** **int** compare(String s1, String s2) {

**if** (s1 == **null** ^ s2 == **null**) {

**return** (s1 == **null**) ? -1 : 1;

} **else** **if** (s1 == **null** && s2 == **null**) {

**return** 0;

}

**return** s1.compareToIgnoreCase(s2);

}

});

System.***out***.println("Sorted List: " + kidsName);

}

Output

Unsorted List: [Robb, Sansa, John, null, Arya]

Sorted List: [null, Arya, John, Robb, Sansa]

That’s all on sort() method. In next post we will see subList method. subList method returns a sub portion of main list given for fromIndex to toIndex.