Collections class sort method.

sort() method is used to sort List of elements. sort() method is overloaded method. Below are the two overloaded sort() methods:

1. sort(List<T> list)
2. sort(List<T> list, Comparator<? **super** T> c)

In this post we will see both the versions of sort() method.

First we will see the **sort(List<T> list)**. This method accepts the List<T>. It sorts the list in ascending order according to Comparable interface’s natural ordering of elements. All elements in list must implement Comparable and they must be mutually comparable that is o1.compareTo(o2) must not throw ClassCastException.

Below is the demo code for Collections.sort(list) method.

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** ListSort {

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

ListSort lsc = **new** ListSort();

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

lsc.sortList(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("Rickon");

names.add("Brandon");

**return** names;

}

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

System.***out***.println("Before Sort: "+names);

Collections.*sort*(names);

System.***out***.println("After Sort: "+names);

}

}

Output:

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

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

If the see the output it is sorted lexical order or alphabetical order. Remember String class implements the Comparable<T> interface so the sorting takes place.

Now let implement the Comparable interface in Person class and sort it.

Below is the Person class that implements the Comparable<T> interface.

**public** **class** Person **implements Comparable<Person>**{

**private** **int** age;

**private** String name;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**/\*\***

**\* compareTo(T o) is used as natural comparison method where**

**\* comparison is done on lexical order on Person's name.**

**\* \*/**

**@Override**

**public int compareTo(Person person) {**

**return this.getName().compareTo(person.getName());**

**}**

@Override

**public** String toString() {

**return** **new** StringBuilder()

.append("{")

.append(**this**.getName())

.append(" ")

.append(**this**.getAge())

.append("}")

.toString();

}

}

Below is Driver class that builts a list and sorts it.

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** PersonComparable {

**public** **static** List<Person> personList() {

List<Person> list = **new** ArrayList<Person>();

list.add(**new** Person("Eddard", 55));

list.add(**new** Person("Rob", 23));

list.add(**new** Person("Joffery", 21));

list.add(**new** Person("Sansa", 19));

list.add(**new** Person("Rickon", 7));

list.add(**new** Person("Brandon", 9));

**return** list;

}

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

List<Person> people = *personList*();

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

**Collections.*sort*(people);**

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

}

}

Output:

Before sort: [{Eddard 55},

{Rob 23},

{Joffery 21},

{Sansa 19},

{Rickon 7},

{Brandon 9}]

After sort: [{Brandon 9},

{Eddard 55},

{Joffery 21},

{Rickon 7},

{Rob 23},

{Sansa 19}]

That’s all on Collections.sort(list). In next post we will see how to implement Comparator<T> for sorting the objects using different ordering.

Notice if there is null in List<T> then Collections.sort(list) will throw NullPointerException. So this is unsafe practice. We can write our own Comparator to avoid NullPointerException.