Comparator Interface

In previous post we saw about Comparable<T> interface. Comparator<T> interface is in overloaded version of Collections.sort() that takes list and Comparator.

Comparator can be passed to sort method Collections.sort(list, Comparator) or Arrays.sort(Object[], Comparator). The Comparator Interface gives you the ability to sort a collection in different ways. Also another thing is that while using Comparator interface for different ways of sorting you do not need to modify the class itself. This is big advantage over the Comparable interface.

Let us now implement sorting by Comparator Interface.

We will use People class and sort the instances by name and age.

**package** org.collections;

**public** **class** People{

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

**private** String name;

**public** People(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;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (obj **instanceof** Person) {

Person o = (Person) obj;

**if** (o.getAge() == **this**.getAge() && o.getName().equals(**this**.getName())) {

**return** **true**;

}

**return** **false**;

}

**return** **false**;

}

@Override

**public** **int** hashCode() {

StringBuilder sb = **new** StringBuilder();

sb.append(**this**.getAge()).append(**this**.getName());

**return** sb.hashCode();

}

@Override

**public** String toString() {

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

.append("{")

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

.append(" ")

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

.append("}")

.toString();

}

}

This is the dummy data

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

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

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

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

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

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

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

The list looks like this

Before Sort [{Eddard 55}, {Rob 23}, {Joffery 21}, {Sansa 19}, {Rickon 7}, {Brandon 9}]

Implement the Comparator<People> interface as anonymous inner class and sort by name.

Collections.*sort*(list, **new** Comparator<People>() {

**public** **int** compare(People p1, People p2) {

**return** p1.getName().compareTo(p2.getName());

}

});

After above sort

Sort by Name [{Brandon 9}, {Eddard 55}, {Joffery 21}, {Rickon 7}, {Rob 23}, {Sansa 19}]

Implement Comparator<People> interface as anonymous inner class and sort by age.

Collections.*sort*(list, **new** Comparator<People>() {

**public** **int** compare(People p1, People p2) {

**if** (p1.getAge() == p2.getAge()) {

**return** 0;

} **else** **if** (p1.getAge() < p2.getAge()) {

**return** -1;

} **else** {

**return** 1;

}

}

});

After above sort

Sort by Age [{Rickon 7}, {Brandon 9}, {Sansa 19}, {Joffery 21}, {Rob 23}, {Eddard 55}]

Now we will write the Comparator<> into the method and return Comparator<> object.

Below method returns Comparator<> that will sort instances by name.

**public** **static** Comparator<People> sortByName() {

Comparator<People> nameSorter = **new** Comparator<People>() {

@Override

**public** **int** compare(People p1, People p2) {

**return** p1.getName().compareTo(p2.getName());

}

};

**return** nameSorter;

}

Collections.*sort*(list, *sortByName*());

Same we can write method that returns Comparator<> object that will sort by age.

**public** **static** Comparator<People> sortByAge() {

Comparator<People> ageSorter = **new** Comparator<People>() {

@Override

**public** **int** compare(People p1, People p2) {

**if** (p1.getAge() == p2.getAge()) {

**return** 0;

} **else** **if** (p1.getAge() < p2.getAge()) {

**return** -1;

} **else** {

**return** 1;

}

}

};

**return** ageSorter;

}

We will see how to use Comparator<> in Java 8 and Difference between Comparable<> and Comparator<> interface in next post.