Collections min method with Comparator

Overloaded version of min(..) method accepts the Collection and Comparator. All elements must be comparable i.e. e1.compareTo(e2) must not throw ClassCastException for any elements in collection.

We saw [min method with Comparable<T> interface](http://data-structure-learning.blogspot.com/2015/06/collections-min-method-with-comparable.html) in previous post.

min(..) 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) and [addAll(..)](http://data-structure-learning.blogspot.com/2015/05/collections-class-addall-method.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.

In this post we will see How to use min method with Comparator<T> interface.

Let us use a People class.

**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();

}

}

Now we will write Comparator as inner class.

**public** **static** **void** minComparatorDemo(){

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));

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

People p = **Collections.*min*(list, new Comparator<People>() {**

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

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

**}**

**});**

System.***out***.println("min by Name " + p);

p = **Collections.*min*(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;**

**}**

**}**

**});**

System.***out***.println("min by Age " + p);

}

Output:

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

min by Name {Brandon 9}

min by Age {Rickon 7}