Java Comparator interface

**Java Comparator interface** is used to order the objects of a user-defined class.

This interface is found in java.util package and contains 2 methods compare(Object obj1,Object obj2) and equals(Object element).

It provides multiple sorting sequences, i.e., you can sort the elements on the basis of any data member, for example, rollno, name, age or anything else.

Methods of Java Comparator Interface

|  |  |
| --- | --- |
| **Method** | **Description** |
| public int compare(Object obj1, Object obj2) | It compares the first object with the second object. |
| public boolean equals(Object obj) | It is used to compare the current object with the specified object. |
| public boolean equals(Object obj) | It is used to compare the current object with the specified object. |

Collections class

**Collections** class provides static methods for sorting the elements of a collection. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements also.

Method of Collections class for sorting List elements

**public void sort(List list, Comparator c):** is used to sort the elements of List by the given Comparator.

Java Comparator Example (Non-generic Old Style)

Let's see the example of sorting the elements of List on the basis of age and name. In this example, we have created 4 java classes:

1. Student.java
2. AgeComparator.java
3. NameComparator.java
4. Simple.java

**Student.java**

This class contains three fields rollno, name and age and a parameterized constructor.

1. **class** Student{
2. **int** rollno;
3. String name;
4. **int** age;
5. Student(**int** rollno,String name,**int** age){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.age=age;
9. }
10. }

**AgeComparator.java**

This class defines comparison logic based on the age. If the age of the first object is greater than the second, we are returning a positive value. It can be anyone such as 1, 2, 10. If the age of the first object is less than the second object, we are returning a negative value, it can be any negative value, and if the age of both objects is equal, we are returning 0.

1. **import** java.util.\*;
2. **class** AgeComparator **implements** Comparator{
3. **public** **int** compare(Object o1,Object o2){
4. Student s1=(Student)o1;
5. Student s2=(Student)o2;
7. **if**(s1.age==s2.age)
8. **return** 0;
9. **else** **if**(s1.age>s2.age)
10. **return** 1;
11. **else**
12. **return** -1;
13. }
14. }

**NameComparator.java**

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

1. **import** java.util.\*;
2. **class** NameComparator **implements** Comparator{
3. **public** **int** compare(Object o1,Object o2){
4. Student s1=(Student)o1;
5. Student s2=(Student)o2;
7. **return** s1.name.compareTo(s2.name);
8. }
9. }

**Simple.java**

In this class, we are printing the values of the object by sorting on the basis of name and age.

1. **import** java.util.\*;
2. **import** java.io.\*;
4. **class** Simple{
5. **public** **static** **void** main(String args[]){
7. ArrayList al=**new** ArrayList();
8. al.add(**new** Student(101,"Vijay",23));
9. al.add(**new** Student(106,"Ajay",27));
10. al.add(**new** Student(105,"Jai",21));
12. System.out.println("Sorting by Name");
14. Collections.sort(al,**new** NameComparator());
15. Iterator itr=al.iterator();
16. **while**(itr.hasNext()){
17. Student st=(Student)itr.next();
18. System.out.println(st.rollno+" "+st.name+" "+st.age);
19. }
21. System.out.println("Sorting by age");
23. Collections.sort(al,**new** AgeComparator());
24. Iterator itr2=al.iterator();
25. **while**(itr2.hasNext()){
26. Student st=(Student)itr2.next();
27. System.out.println(st.rollno+" "+st.name+" "+st.age);
28. }

31. }
32. }

Sorting by Name

106 Ajay 27

105 Jai 21

101 Vijay 23

Sorting by age

105 Jai 21

101 Vijay 23

106 Ajay 27

Java Comparator Example (Generic)

**Student.java**

1. **class** Student{
2. **int** rollno;
3. String name;
4. **int** age;
5. Student(**int** rollno,String name,**int** age){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.age=age;
9. }
10. }

**AgeComparator.java**

1. **import** java.util.\*;
2. **class** AgeComparator **implements** Comparator<Student>{
3. **public** **int** compare(Student s1,Student s2){
4. **if**(s1.age==s2.age)
5. **return** 0;
6. **else** **if**(s1.age>s2.age)
7. **return** 1;
8. **else**
9. **return** -1;
10. }
11. }

**NameComparator.java**

This class provides comparison logic based on the name. In such case, we are using the compareTo() method of String class, which internally provides the comparison logic.

1. **import** java.util.\*;
2. **class** NameComparator **implements** Comparator<Student>{
3. **public** **int** compare(Student s1,Student s2){
4. **return** s1.name.compareTo(s2.name);
5. }
6. }

**Simple.java**

In this class, we are printing the values of the object by sorting on the basis of name and age.

1. **import** java.util.\*;
2. **import** java.io.\*;
3. **class** Simple{
4. **public** **static** **void** main(String args[]){
6. ArrayList<Student> al=**new** ArrayList<Student>();
7. al.add(**new** Student(101,"Vijay",23));
8. al.add(**new** Student(106,"Ajay",27));
9. al.add(**new** Student(105,"Jai",21));
11. System.out.println("Sorting by Name");
13. Collections.sort(al,**new** NameComparator());
14. **for**(Student st: al){
15. System.out.println(st.rollno+" "+st.name+" "+st.age);
16. }
18. System.out.println("Sorting by age");
20. Collections.sort(al,**new** AgeComparator());
21. **for**(Student st: al){
22. System.out.println(st.rollno+" "+st.name+" "+st.age);
23. }
24. }
25. }

Sorting by Name

106 Ajay 27

105 Jai 21

101 Vijay 23

Sorting by age

105 Jai 21

101 Vijay 23

106 Ajay 27

Java 8 Comparator interface

Java 8 Comparator interface is a functional interface that contains only one abstract method. Now, we can use the Comparator interface as the assignment target for a lambda expression or method reference.

Methods of Java 8 Comparator Interface

|  |
| --- |
|  |

|  |  |
| --- | --- |
| **Method** | **Description** |
| int compare(T o1, T o2) | It compares the first object with second object. |
| static <T,U extends Comparable<? super U>> Comparator<T> comparing(Function<? super T,? extends U> keyExtractor) | It accepts a function that extracts a Comparable sort key  from a type T, and returns a Comparator that compares by  that sort key. |
| static <T,U> Comparator<T> comparing(Function<? super T,? extends U> keyExtractor, Comparator<? super U> keyComparator) | It accepts a function that extracts a sort key from a type T,  and returns a Comparator that compares by that sort key  using the specified Comparator. |
| static <T> Comparator<T> comparingDouble(ToDoubleFunction<? super T> keyExtractor) | It accepts a function that extracts a double sort key from  a type T, and returns a Comparator that compares by  that sort key. |
| static <T> Comparator<T> comparingInt(ToIntFunction<? super T> keyExtractor) | It accepts a function that extracts an int sort key from a type T  , and returns a Comparator that compares by that sort key. |
| static <T> Comparator<T> comparingLong(ToLongFunction<? super T> keyExtractor) | It accepts a function that extracts a long sort key from a type T,  and returns a Comparator that compares by that sort key. |
| boolean equals(Object obj) | It is used to compare the current object with the specified object. |
| static <T extends Comparable<? super T>> Comparator<T> naturalOrder() | It returns a comparator that compares Comparable objects in natural order. |
| static <T> Comparator<T> nullsFirst(Comparator<? super T> comparator) | It returns a comparator that treats null to be less than non-null  elements. |
| static <T> Comparator<T> nullsLast(Comparator<? super T> comparator) | It returns a comparator that treats null to be greater than  non-null elements. |
| default Comparator<T> reversed() | It returns comparator that contains reverse ordering of  the provided comparator. |
| static <T extends Comparable<? super T>> Comparator<T> reverseOrder() | It returns comparator that contains reverse of natural ordering. |
| default Comparator<T> thenComparing(Comparator<? super T> other) | It returns a lexicographic-order comparator with another  comparator. |
| default <U extends Comparable<? super U>> Comparator<T> thenComparing(Function<? super T,? extends U> keyExtractor) | It returns a lexicographic-order comparator with a  function that extracts a Comparable sort key. |
| default <U> Comparator<T> thenComparing(Function<? super T,? extends U> keyExtractor, Comparator<? super U> keyComparator) | It returns a lexicographic-order comparator with a  function that extracts a key to be compared with the given  Comparator. |
| default Comparator<T> thenComparingDouble(ToDoubleFunction<? super T> keyExtractor) | It returns a lexicographic-order comparator with a  function that extracts a double sort key. |
| default Comparator<T> thenComparingInt(ToIntFunction<? super T> keyExtractor) | It returns a lexicographic-order comparator with a  function that extracts a int sort key. |
| default Comparator<T> thenComparingLong(ToLongFunction<? super T> keyExtractor) | It returns a lexicographic-order comparator with a  function that extracts a long sort key. |

Java 8 Comparator Example

Let's see the example of sorting the elements of List on the basis of age and name.

*File: Student.java*

1. **class** Student {
2. **int** rollno;
3. String name;
4. **int** age;
5. Student(**int** rollno,String name,**int** age){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.age=age;
9. }
11. **public** **int** getRollno() {
12. **return** rollno;
13. }
15. **public** **void** setRollno(**int** rollno) {
16. **this**.rollno = rollno;
17. }
19. **public** String getName() {
20. **return** name;
21. }
23. **public** **void** setName(String name) {
24. **this**.name = name;
25. }
27. **public** **int** getAge() {
28. **return** age;
29. }
31. **public** **void** setAge(**int** age) {
32. **this**.age = age;
33. }
35. }

*File: TestSort1.java*

1. **import** java.util.\*;
2. **public** **class** TestSort1{
3. **public** **static** **void** main(String args[]){
4. ArrayList<Student> al=**new** ArrayList<Student>();
5. al.add(**new** Student(101,"Vijay",23));
6. al.add(**new** Student(106,"Ajay",27));
7. al.add(**new** Student(105,"Jai",21));
8. /Sorting elements on the basis of name
9. Comparator<Student> cm1=Comparator.comparing(Student::getName);
10. Collections.sort(al,cm1);
11. System.out.println("Sorting by Name");
12. **for**(Student st: al){
13. System.out.println(st.rollno+" "+st.name+" "+st.age);
14. }
15. //Sorting elements on the basis of age
16. Comparator<Student> cm2=Comparator.comparing(Student::getAge);
17. Collections.sort(al,cm2);
18. System.out.println("Sorting by Age");
19. **for**(Student st: al){
20. System.out.println(st.rollno+" "+st.name+" "+st.age);
21. }
22. }
23. }

Sorting by Name

106 Ajay 27

105 Jai 21

101 Vijay 23

Sorting by Age

105 Jai 21

101 Vijay 23

106 Ajay 27

Java 8 Comparator Example: nullsFirst() and nullsLast() method

Here, we sort the list of elements that also contains null.

*File: Student.java*

1. **class** Student {
2. **int** rollno;
3. String name;
4. **int** age;
5. Student(**int** rollno,String name,**int** age){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.age=age;
9. }
10. **public** **int** getRollno() {
11. **return** rollno;
12. }
13. **public** **void** setRollno(**int** rollno) {
14. **this**.rollno = rollno;
15. }
16. **public** String getName() {
17. **return** name;
18. }
20. **public** **void** setName(String name) {
21. **this**.name = name;
22. }
24. **public** **int** getAge() {
25. **return** age;
26. }
27. **public** **void** setAge(**int** age) {
28. **this**.age = age;
29. }
30. }

*File: TestSort2.java*

1. **import** java.util.\*;
2. **public** **class** TestSort2{
3. **public** **static** **void** main(String args[]){
4. ArrayList<Student> al=**new** ArrayList<Student>();
5. al.add(**new** Student(101,"Vijay",23));
6. al.add(**new** Student(106,"Ajay",27));
7. al.add(**new** Student(105,**null**,21));
8. Comparator<Student> cm1=Comparator.comparing(Student::getName,Comparator.nullsFirst(String::compareTo));
9. Collections.sort(al,cm1);
10. System.out.println("Considers null to be less than non-null");
11. **for**(Student st: al){
12. System.out.println(st.rollno+" "+st.name+" "+st.age);
13. }
14. Comparator<Student> cm2=Comparator.comparing(Student::getName,Comparator.nullsLast(String::compareTo));
15. Collections.sort(al,cm2);
16. System.out.println("Considers null to be greater than non-null");
17. **for**(Student st: al){
18. System.out.println(st.rollno+" "+st.name+" "+st.age);
19. }
20. }
21. }

Considers null to be less than non-null

105 null 21

106 Ajay 27

101 Vijay 23

Considers null to be greater than non-null

106 Ajay 27

101 Vijay 23

105 null 21