HowToDoInJava

Sorting Streams in Java

Last Updated: March 4, 2022 By: Lokesh Gupta Java 8 Java Sorting, Java Stream Basics

Learn to **sort streams** of numbers, strings and custom types in ascending (**natural order**) and descending orders (**reverse order**) in Java.

Table Of Contents

1. Basics of Sorting the Streams

The *Stream* interface provides the *sorted()* method that returns a stream consisting of the elements of a given stream, sorted according to the natural order. It is an overloaded method:

- `Stream sorted()`: sorted according to natural order.
- `Stream sorted(comparator)`: sorted according to the provided Comparator.

Note that both methods are *intermediate operations* so we still need to call a terminal operation to trigger the sorting.

Pseudo Code

```
Stream<Person> unsortedStream;

//Default Ordering

List<Person> sortedList = unsortedStream.sorted()
    .collect(Collectors.toList());

//Order by First Name
```

```
List<Person> sortedList = unsortedStream.sorted(Person::firstName)
    .collect(Collectors.toList());
```

2. Sorting Custom Types

For demonstration purposes, we are using the custom class `Person`. This class has only three fields: id, first name and last name.

By default, two persons are considered equal if their `id` *is equal.*

Default sorting is by id field

```
import java.util.Objects;

public class Person implements Comparable<Person> {

    private Integer id;
    private String fname;
    private String lname;

    //Constructor, Setters and Getters are hidden for brevity

    @Override
    public int compareTo(Person p) {
        return this.getId().compareTo(p.getId());
    }
}
```

2.1. Default Sorting

By default, the *sorted()* method uses the *Comparable.compareTo()* method implemented by the `Person` class.

As *Person* class compares the instances using the value of *id* field, so when we sort the stream of *Person* instances – we get the instances **sorted by *id***. The default sorting is in the **natural order**.

Natural Sorting

```
Stream<Person> personStream = getPersonStream();

// Ascending Order
personStream.sorted()
            .forEach(System.out::println);
```

Output

```
Person [id=1, fname=Lokesh, lname=Gupta]
Person [id=2, fname=Lokesh, lname=Gupta]
Person [id=3, fname=Brian, lname=Clooney]
Person [id=4, fname=Brian, lname=Clooney]
Person [id=5, fname=Lokesh, lname=Gupta]
```

The same is true for reverse sorting as well. we can sort the *Person* instances in **reverse order** by passing the reverse comparator obtained from *Comparator.reverseOrder()* method into the *sorted()* method.

Reverse Ordering

```
Stream<Person> personStream = getPersonStream();

// Reverse Order
personStream.sorted(Comparator.reverseOrder())
            .forEach(System.out::println);
```

Output

```
Person [id=6, fname=Alex, lname=Kolen]
Person [id=5, fname=Lokesh, lname=Gupta]
Person [id=4, fname=Brian, lname=Clooney]
Person [id=3, fname=Brian, lname=Clooney]
Person [id=2, fname=Lokesh, lname=Gupta]
Person [id=1, fname=Lokesh, lname=Gupta]
```

2.2. Custom Sorting

What if we want to **sort the *Person* instances by their first name**. The default sort does not support it, so we need to create our custom comparator.

FirstNameSorter.java

```
import java.util.Comparator;
import com.howtodoinjava.core.streams.Person;

public class FirstNameSorter implements Comparator<Person>{

    @Override
    public int compare(Person p1, Person p2) {
        if(p1.getFname() == null || p2.getFname() == null) {
            throw new IllegalArgumentException("Unnamed Person found in the system");
        }
       return p1.getFname().compareToIgnoreCase(p2.getFname());
    }
}
```

Now pass the *FirstNameSorter* instance to the *sorted()* method. This time, sorting will use the *compare()* method written in *FirstNameSorter*.

Sorting with first name

```
List<Person> sortedList = personStream.sorted(new FirstNameSorter())
    .collect(Collectors.toList());

sortedList.forEach(System.out::println);
```

Output

```
Person [id=6, fname=Alex, lname=Kolen]
Person [id=4, fname=Brian, lname=Clooney]
Person [id=3, fname=Brian, lname=Clooney]
Person [id=1, fname=Lokesh, lname=Gupta]
Person [id=5, fname=Lokesh, lname=Gupta]
Person [id=2, fname=Lokesh, lname=Gupta]
```

Similarly, to **sort the instances by the first name in reverse order**, we can reverse any comparator using its *reverse()* method.

Reverse Sorting by First Name

```
List<Person> reverseSortedList = personStream.sorted( new FirstNameSorter().reversed() )
                                        .collect(Collectors.toList());

reverseSortedList.forEach(System.out::println);
```

2.3. Class cannot be cast to class java.lang.Comparable

Please note that if our custom class *Person* does not implement the Comparable interface then we will get the `ClassCastException` in runtime while doing the natural sorting.

```
Exception in thread "main" java.lang.ClassCastException: class com.howtodoinjava.core.streams
 cannot be cast to class java.lang.Comparable (com.howtodoinjava.core.streams.sort.Person is
 module of loader 'app'; java.lang.Comparable is in module java.base of loader 'bootstrap')
    at java.base/java.util.Comparators $NaturalOrderComparator.compare(Comparators.java:47)
    at java.base/java.util.TimSort. countRunAndMakeAscending(TimSort.java:355)
    at java.base/java.util.TimSort. sort(TimSort.java:220)
    at java.base/java.util.Arrays. sort(Arrays.java:1307)
```

3. Sorting Stream of Numbers

3.1. Ascending Order

Java programs to sort a stream of numbers using **Stream.sorted()** method.

Ascending sort example

```
import java.util.stream.Stream;

public class Main
{
    public static void main(String[] args)
    {
        Stream<Integer> numStream = Stream.of(1,3,5,4,2);

        numStream.sorted()
                 .forEach(System.out::println);
    }
}
```

Program output.

Output

```
1
2
3
4
5
```

3.2. Descending Order

To sort in reverse order, use Comparator.reverseOrder() in `sorted()` method.

Descending sort example

```
import java.util.Comparator;
import java.util.stream.Stream;

public class Main
{
    public static void main(String[] args)
    {
        Stream<Integer> numStream = Stream.of(1,3,5,4,2);

        numStream.sorted( Comparator.reverseOrder() )
                 .forEach(System.out::println);
    }
}
```

4. Sorting Stream of Strings

Java programs to sort a stream of strings using **Stream.sorted()** method in ascending and descending order.

Sort stream of strings

```
Stream<String> wordStream = Stream.of("A","C","E","B","D");

wordStream.sorted()                                 //ascending
          .forEach(System.out::println);

wordStream.sorted( Comparator.reverseOrder() )      //descending
          .forEach(System.out::println);
```

Program output.

Output

```
A B C D E

E D C B A
```

Happy Learning !!

Sourcecode on Github

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Boxed Streams in Java
2. Using 'if-else' Conditions with Java Streams
3. Creating Infinite Streams in Java
4. Applying Multiple Conditions on Java Streams
5. Finding Max and Min from List using Streams
6. Sorting a Stream by Multiple Fields in Java
7. Java Streams API
8. Creating Streams in Java
9. Primitive Type Streams in Java
0. Java – Sorting Array of Strings in Alphabetical Order

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...