

If you’ve been working with Java for some time, then you’ve probably encountered a scenario where you need to sort the elements in a collection.

If your collection contains a wrapper class object then the sorting is very easy. Since all the wrapper classes implement the `Comparable` interface, you can directly use `Collections.sort()` to sort your collection.

However, if your collection contains a custom class object then you need to provide the logic to sort your object. In this lesson, we will look at an example in which we will sort a list of `Person` class objects using a comparator. Then, we will write a program to do the same task using lambdas.

## Comparator example using anonymous class#

First, we will create a `Person` class.

```
1 public class Person {
2
3     private String name;
4     private int age;
5     private String country;
6
7     public Person(String name, int age, String country) {
8         this.name = name;
9         this.age = age;
10        this.country = country;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public int getAge() {
18        return age;
19    }
20
21    public String getCountry() {
22        return country;
23    }
24 }
```

Now, we have a `PersonService` class. It has a `getPersons(List<Person> persons)` method. It takes a list of person objects as input and returns a list of person object in sorted order.

In this method, we are creating an anonymous comparator, which sorts the `Person` objects on the basis of name.

```
1 import java.util.Collections;
2 import java.util.Comparator;
3 import java.util.List;
4
5 public class PersonService {
6
7     public static List<Person> getPersons(List<Person> persons){
8         // Created an anonymous Comparator, which sorts the Person object on the basis of Person name.
9         Collections.sort(persons, new Comparator<Person>() {
10             @Override
11             public int compare(Person p1, Person p2) {
12                 return p1.getName().compareTo(p2.getName());
13             }
14         });
15         return persons;
16     }
17 }
18
```

Finally, we have a `PersonMain` class that runs our logic.

PersonMain.java

PersonService.java

Person.java

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class PersonMain {
5
6     public static void main(String args[]){
7         List<Person> persons = new ArrayList<>();
8         persons.add(new Person("John" , 23 , "USA"));
9         persons.add(new Person("Carl" , 23 , "Australia"));
10        persons.add(new Person("Amit" , 23 , "India"));
11        persons.add(new Person("Vikram" , 23 , "Bhutan"));
12        persons.add(new Person("Kane" , 23 , "Brazil"));
13        // Calling getPerson() method which will return the List of Person in sorted order.
14        List<Person> sortedPersons = PersonService.getPersons(persons);
15
16        System.out.println("Persons after sorting");
17        // Printing the name of each person.
18        for(Person person : sortedPersons){
19            System.out.println("Person Name : " + person.getName());
20        }
21    }
22 }
23
```

Run

Save

Reset

If you look at the `Comparator` interface, you notice that it is a functional interface. It has only one abstract method called `compare()`. This makes it a perfect candidate to be used in lambdas.

## Comparator example using a lambda expression#

Now, let’s see how we can write the same logic using a lambda expression. As discussed in the previous lesson, when writing lambdas, we only need to consider the input parameters and the method body.

Below is the signature of the `compare()` method.

```
int compare(T o1, T o2)
```

It takes two parameters as input and returns an `int`.

Let’s start constructing the lambda expression:

The structure of lambda will be like:

```
(p1, p2) -> {};
```

Here, `p1` and `p2` are the two input parameters. We can name them anything.

Now, we will add the body.

```
(p1, p2) -> p1.getName().compareTo(p2.getName());
```

So, this is the lambda expression for sorting the `Person` objects based on name.

You can see how easy and concise it is to write code with lambdas instead of using anonymous classes.

PersonService.java

PersonMain.java

Person.java

```
1 import java.util.Collections;
2 import java.util.Comparator;
3 import java.util.List;
4
5 public class PersonService {
6
7     public static List<Person> getPersons(List<Person> persons) {
8         // Instead of creating an anonymous class, we have provided a lambda expression.
9         Collections.sort(persons, (p1, p2) -> p1.getName().compareTo(p2.getName()));
10        return persons;
11    }
12 }
```

Run

Save

Reset