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.

```
public class Person {

    private String name;
    private int age;
    private String country;

    public Person(String name, int age, String country) {
        this.name = name;
        this.age = age;
        this.country = country;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getCountry() {
        return country;
    }
}
```

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.

```
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class PersonService {

    public static List<Person> getPersons(List<Person> persons){
        // Created an anonymous Comparator, which sorts the Person object on the basis of Person name.
        Collections.sort(persons, new Comparator<Person>() {
            @Override
            public int compare(Person p1, Person p2) {
                return p1.getName().compareTo(p2.getName());
            }
        });
        return persons;
    }
}

```

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

PersonMain.java
PersonService.java
Person.java

```
import java.util.ArrayList;
import java.util.List;

public class PersonMain {

    public static void main(String args[]){
        List<Person> persons = new ArrayList<>();
        persons.add(new Person("John" , 23 , "USA"));
        persons.add(new Person("Carl" , 23 , "Australia"));
        persons.add(new Person("Amit" , 23 , "India"));
        persons.add(new Person("Vikram" , 23 , "Bhutan"));
        persons.add(new Person("Kane" , 23 , "Brazil"));
        // Calling getPerson() method which will return the List of Person in
        List<Person> sortedPersons = PersonService.getPersons(persons);

        System.out.println("Persons after sorting");
        // Printing the name of each person.
        for(Person person : sortedPersons){
            System.out.println("Person Name : " + person.getName());
        }
    }
}
```

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

```
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class PersonService {

    public static List<Person> getPersons(List<Person> persons) {
        // Instead of creating an anonymous class, we have provided a lambda
        Collections.sort(persons, (p1, p2) -> p1.getName().compareTo(p2.getNa
        return persons;
    }
}
```

Run Save Reset

In the next lesson, you will learn about the `Predicate` functional interface.