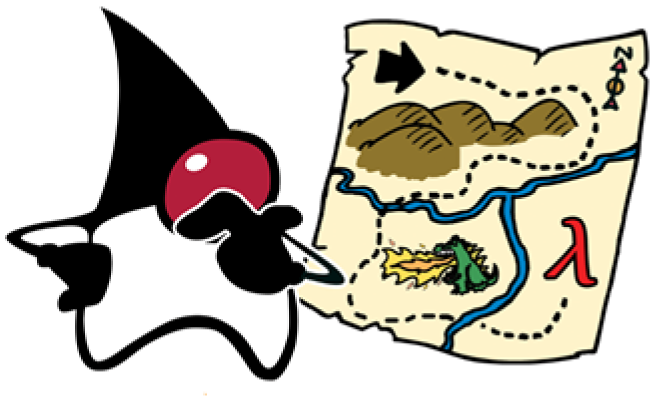
Search...

Java 8 Lambda : Comparator example

By mkyong | Last updated: August 5, 2015
Viewed: 869,113

Tags: comparator | java8 | lambda | sorting



In this example, we will show you how to use Java 8 Lambda expression to write a `Comparator` to sort a List.

1. Classic `Comparator` example.

```
	Comparator<Developer> byName = new Comparator<Developer>() {
		@Override
		public int compare(Developer o1, Developer o2) {
			return o1.getName().compareTo(o2.getName());
		}
	};
```

2. Lambda expression equivalent.

Privacy

```
        Comparator<Developer> byName =
                (Developer o1, Developer o2)->o1.getName().compareTo(o2.getNa
```

1. Sort without Lambda

Example to compare the `Developer` objects using their age. Normally, you use `Collections.sort` and pass an anonymous `Comparator` class like this :

TestSorting.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class TestSorting {

        public static void main(String[] args) {

                List<Developer> listDevs = getDevelopers();

                System.out.println("Before Sort");
                for (Developer developer : listDevs) {
                        System.out.println(developer);
                }

                //sort by age
                Collections.sort(listDevs, new Comparator<Developer>() {
                        @Override
                        public int compare(Developer o1, Developer o2) {
                                return o1.getAge() - o2.getAge();
                        }
                });

                System.out.println("After Sort");
                for (Developer developer : listDevs) {
                        System.out.println(developer);
                }
```

Privacy

```
	}

	private static List<Developer> getDevelopers() {

		List<Developer> result = new ArrayList<Developer>();

		result.add(new Developer("mkyong", new BigDecimal("70000"), 3
		result.add(new Developer("alvin", new BigDecimal("80000"), 20
		result.add(new Developer("jason", new BigDecimal("100000"), 1
		result.add(new Developer("iris", new BigDecimal("170000"), 55

		return result;

	}

}
```

Output

```
Before Sort
Developer [name=mkyong, salary=70000, age=33]
Developer [name=alvin, salary=80000, age=20]
Developer [name=jason, salary=100000, age=10]
Developer [name=iris, salary=170000, age=55]

After Sort
Developer [name=jason, salary=100000, age=10]
Developer [name=alvin, salary=80000, age=20]
Developer [name=mkyong, salary=70000, age=33]
Developer [name=iris, salary=170000, age=55]
```

When the sorting requirement is changed, you just pass in another new anonymous `Comparator` class :

```
	//sort by age
	Collections.sort(listDevs, new Comparator<Developer>() {
		@Override
		public int compare(Developer o1, Developer o2) {
			return o1.getAge() - o2.getAge();
		}
	});
```

Privacy

```
		//sort by name
		Collections.sort(listDevs, new Comparator<Developer>() {
			@Override
			public int compare(Developer o1, Developer o2) {
				return o1.getName().compareTo(o2.getName());
			}
		});

		//sort by salary
		Collections.sort(listDevs, new Comparator<Developer>() {
			@Override
			public int compare(Developer o1, Developer o2) {
				return o1.getSalary().compareTo(o2.getSalary());
			}
		});
```

It works, but, do you think it is a bit weird to create a class just because you want to change a single line of code?

2. Sort with Lambda

In Java 8, the `List` interface is supports the `sort` method directly, no need to use `Collections.sort` anymore.

```
		//List.sort() since Java 8
		listDevs.sort(new Comparator<Developer>() {
			@Override
			public int compare(Developer o1, Developer o2) {
				return o2.getAge() - o1.getAge();
			}
		});
```

Lambda expression example :

```
TestSorting.java
```

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

public class TestSorting {

        public static void main(String[] args) {

                List<Developer> listDevs = getDevelopers();

                System.out.println("Before Sort");
                for (Developer developer : listDevs) {
                        System.out.println(developer);
                }

                System.out.println("After Sort");

                //lambda here!
                listDevs.sort((Developer o1, Developer o2)->o1.getAge()-o2.ge

                //java 8 only, lambda also, to print the List
                listDevs.forEach((developer)->System.out.println(developer));
        }

        private static List<Developer> getDevelopers() {

                List<Developer> result = new ArrayList<Developer>();

                result.add(new Developer("mkyong", new BigDecimal("70000"), 3
                result.add(new Developer("alvin", new BigDecimal("80000"), 20
                result.add(new Developer("jason", new BigDecimal("100000"), 1
                result.add(new Developer("iris", new BigDecimal("170000"), 55

                return result;

        }

}
```

Output

```
Before Sort
Developer [name=mkyong, salary=70000, age=33]
Developer [name=alvin, salary=80000, age=20]
Developer [name=jason, salary=100000, age=10]
Developer [name=iris, salary=170000, age=55]
```

Privacy

```
After Sort
Developer [name=jason, salary=100000, age=10]
Developer [name=alvin, salary=80000, age=20]
Developer [name=mkyong, salary=70000, age=33]
Developer [name=iris, salary=170000, age=55]
```

3. More Lambda Examples

3.1 Sort By age

```
		//sort by age
		Collections.sort(listDevs, new Comparator<Developer>() {
			@Override
			public int compare(Developer o1, Developer o2) {
				return o1.getAge() - o2.getAge();
			}
		});

		//lambda
		listDevs.sort((Developer o1, Developer o2)->o1.getAge()-o2.getAge());

		//lambda, valid, parameter type is optional
		listDevs.sort((o1, o2)->o1.getAge()-o2.getAge());
```

3.2 Sort by name

```
		//sort by name
		Collections.sort(listDevs, new Comparator<Developer>() {
			@Override
			public int compare(Developer o1, Developer o2) {
				return o1.getName().compareTo(o2.getName());
			}
		});

		//lambda
		listDevs.sort((Developer o1, Developer o2)->o1.getName().compareTo(o2

		//lambda
		listDevs.sort((o1, o2)->o1.getName().compareTo(o2.getName()));
```

Privacy

3.3 Sort by salary

```
		//sort by salary
		Collections.sort(listDevs, new Comparator<Developer>() {
			@Override
			public int compare(Developer o1, Developer o2) {
				return o1.getSalary().compareTo(o2.getSalary());
			}
		});

		//lambda
		listDevs.sort((Developer o1, Developer o2)->o1.getSalary().compareTo(

		//lambda
		listDevs.sort((o1, o2)->o1.getSalary().compareTo(o2.getSalary()));
```

3.4 Reversed sorting.

3.4.1 Lambda expression to sort a List using their salary.

```
		Comparator<Developer> salaryComparator = (o1, o2)->o1.getSalary().com
		listDevs.sort(salaryComparator);
```

Output

```
	Developer [name=mkyong, salary=70000, age=33]
	Developer [name=alvin, salary=80000, age=20]
	Developer [name=jason, salary=100000, age=10]
	Developer [name=iris, salary=170000, age=55]
```

3.4.2 Lambda expression to sort a List using their salary, reversed order.

```
		Comparator<Developer> salaryComparator = (o1, o2)->o1.getSalary().com
		listDevs.sort(salaryComparator.reversed());
```

Privacy