HowToDoInJava

Collectors teeing() method examples

Last Updated: December 26, 2020 By: Lokesh Gupta Java 12 Java Stream Basics

Learn about **Collectors.teeing()** method (added in Java 12), method syntax and how to apply `teeing()` method in various usecases in Java.

1. Purpose of teeing collector

It is a new static method teeing to **java.util.stream.Collectors** interface which allows to collect using two independent collectors, then merge their results using the supplied **BiFunction**.

Every element passed to the resulting collector is processed by both downstream collectors, then their results are merged using the specified merge function into the final result.

Please note that this function helps in performing a certain task in single steps. We can already perform the given task in two steps if we do not use the `teeing()` function. It's just a helper function which helps in reducing the verbosity.

2. Syntax

```
Method Syntax

/**
* downstream1 - the first downstream collector
* downstream2 - the second downstream collector
* merger - the function which merges two results into the single one
```

```
* returns - a Collector which aggregates the results of two supplied collectors.
*/

public static Collector teeing (Collector downstream1,
                Collector downstream2,
                BiFunction merger);
```

3. Use teeing() to find max and min salaried employees

In this **Collectors.teeing() example**, we have a list of employees. We want to find out employee with maximum salary and employee with minimum salary in single step.

The following java program performs finding max and min operations, then collect both items in a Map.

Main.java

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.HashMap;
import java.util.Optional;
import java.util.stream.Collectors;

public class Main
{
  public static void main(String[] args)
  {
    List<Employee> employeeList = Arrays.asList(
                  new Employee(1, "A", 100),
                  new Employee(2, "B", 200),
                  new Employee(3, "C", 300),
                  new Employee(4, "D", 400));

    HashMap<String, Employee> result = employeeList.stream().collect(
              Collectors.teeing(
                  Collectors.maxBy(Comparator.comparing(Employee::getSalary)),
                  Collectors.minBy(Comparator.comparing(Employee::getSalary)),
                  (e1, e2) -> {
                    HashMap<String, Employee> map = new HashMap();
                    map.put("MAX", e1.get());
                    map.put("MIN", e2.get());
```

```
                    return map;
                }
            ));

    System.out.println(result);
  }
}
```

Program output.

```
Console

C:\BAML\DFCCUI\installs\jdk-12.0.1\bin>java Main.java

{
  MIN=Employee [id=1, name=A, salary=100.0],
  MAX=Employee [id=4, name=D, salary=400.0]
}
```

Here **`Employee`** class is like this.

```
Employee.java

class Employee
{
  private long id;
  private String name;
  private double salary;

  public Employee(long id, String name, double salary) {
    super();
    this.id = id;
    this.name = name;
    this.salary = salary;
  }

  //Getters and setters

  @Override
  public String toString() {
    return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";
  }
}
```

4. Use teeing() to filter items and count them

In this example, we will use the same set of employees. Here, we will find all employees with salary greater than 200, and then we will also count the number of such employees.

Main.java

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.HashMap;
import java.util.Optional;
import java.util.stream.Collectors;

public class Main
{
  public static void main(String[] args)
  {
    List<Employee> employeeList = Arrays.asList(
                    new Employee(1, "A", 100),
                    new Employee(2, "B", 200),
                    new Employee(3, "C", 300),
                    new Employee(4, "D", 400));

    HashMap<String, Object> result = employeeList.stream().collect(
              Collectors.teeing(
                  Collectors.filtering(e -> e.getSalary() > 200, Collectors.toList
                  Collectors.filtering(e -> e.getSalary() > 200, Collectors.counti
                  (list, count) -> {
                    HashMap<String, Object> map = new HashMap();
                    map.put("list", list);
                    map.put("count", count);
                    return map;
                  }
              ));

    System.out.println(result);
  }
}
```

Program output.

Console

```
C:\BAML\DFCCUI\installs\jdk-12.0.1\bin>java Main.java

{
  count=2,
  list=[Employee [id=3, name=C, salary=300.0], Employee [id=4, name=D, salary=400.
}
```

5. Conclusion

Above examples of Collectors.teeing() method are very simple and written for basic understanding. You need to use the function very specific to your own need.

Simply remember that when you need to perform stream operation twice and collect results in two different collectors, consider using **teeing()** method. It will not always fit in the usecase, but it may be useful when it fits in.

Happy Learning !!

Reference : Java Doc

Recommended Reading:

1. Spring Method Security with @PreAuthorize and @Secured
2. Spring static factory-method example

3. Java 8 method reference example
4. Java Static – Variable, Method, Block, Class and Import Statement
5. Java String charAt() method example
6. Java String endsWith() method example
7. Java String concat() method example
8. Java String replace() method example
9. ArrayList remove() method example
0. Python ascii() Method

Leave a Comment

Name *

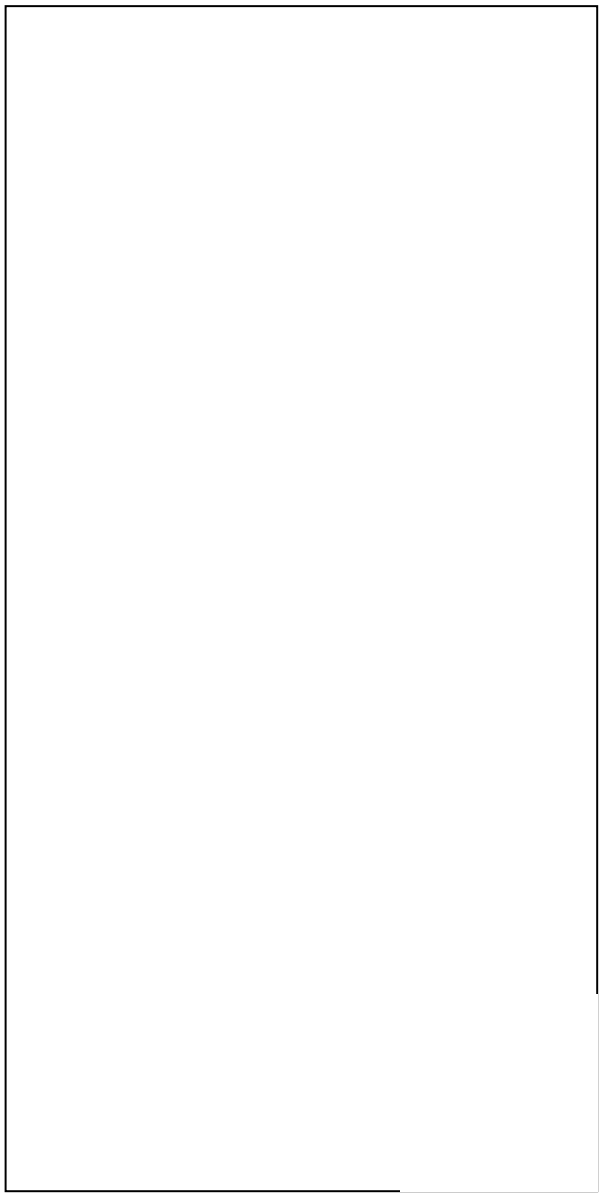
Email *

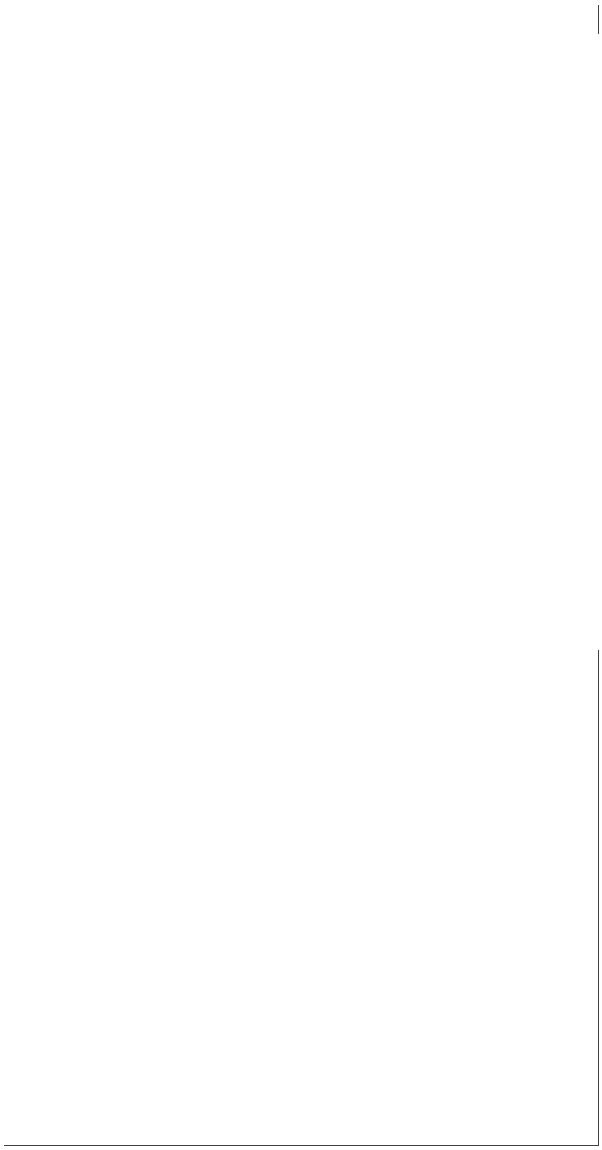
Website

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

Post Comment

Search ...





HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise

Guest Posts

Blogs

REST API Tutorial