HowToDoInJava

Chaining Multiple Predicates in Java

Last Updated: March 10, 2022 | By: Lokesh Gupta | Java 8 | Java Predicate, Java Stream Basics

Learn to combine multiple Predicate instances aka **chained predicates** and perform 'logical AND' and 'logical OR' operations on the Stream *filter()* operation. Chained predicates are useful in filtering the stream items for multiple conditions.

Table Of Contents

1. How to Use Predicates

Predicates are used for filtering the items from a stream. For example, if a have a stream of strings and want to find all the strings starting with '**A**', we can create a *Predicate* using the lambda expression.

Predicate to filter all strings starting with A

```
Predicate<String> startsWithA = s -> s.startsWith("A");
```

Now use this predicate with *Stream.filter()* method.

Using Predicate to Filter Stream Items

```
List<String> list = Arrays.asList("Aa", "Bb", "Cc", "Dd", "Ab", "Bc")

Predicate<String> startsWithA = s -> s.startsWith("A");

List<String> items = list.stream()
  .filter(startsWithA)
  .collect(Collectors.toList());

System.out.println(items);
```

2. Predicate Chain

The first example is of a simple predicate or single condition. In real-world applications, we may be filtering the items on multiple conditions.

2.1. Simple Example

A good way to apply such complex conditions is by combining multiple simple conditions to make one complex condition.

For example, if we want to get *all strings that start with either A or B but it must not contain the letter 'c'*. Lets create the *Predicate* for this:

All strings that start with either A or B but it must not contain the letter c

```
Predicate<String> startsWithA = s -> s.startsWith("A");
Predicate<String> startsWithB = s -> s.startsWith("B");
Predicate<String> notContainsC = s -> !s.contains("c");

Predicate<String> complexPredicate = startsWithA.or(startsWithB)
  .and(notContainsC);
```

Note that for creating the negative conditions, we can use the method **negate()** on the position predicates.

negatedPredicate and notContainsC have the same effect

```
Predicate<String> containsC = s -> s.contains("c");
Predicate<String> negatedPredicate = containsC.negate();

Predicate<String> notContainsC = s -> !s.contains("c");
```

In the above example, `negatedPredicate` and `notContainsC` will have the same effect on the filter() operation.

2.2. The `and()` Method for Logical AND Operation

- The *and()* method returns a **composed predicate** that represents a ***short-circuiting logical AND* of given predicate and another**.
- When evaluating the composed predicate, **if the first predicate is `false`, then the other predicate is not evaluated.**
- Any **exceptions thrown during evaluation of either predicate are relayed to the caller**; if evaluation of first predicate throws an exception, the other predicate will not be evaluated.

In the given example, we are finding all the employees whose *id* is less than 4 and *salary* is greater than 200.

id is less than 4 AND salary is greater than 200

```
List<Employee> employeesList = Arrays.asList(
          new Employee(1, "Alex", 100),
          new Employee(2, "Brian", 200),
          new Employee(3, "Charles", 300),
          new Employee(4, "David", 400),
```

```
            new Employee(5, "Edward", 500),
            new Employee(6, "Frank", 600)
          );

  Predicate<Employee> idLessThan4 = e -> e.getId() < 4;

  Predicate<Employee> salaryGreaterThan200 = e -> e.getSalary() > 200;

  List<Employee> filteredEmployees = employeesList.stream()
      .filter( idLessThan4.and( salaryGreaterThan200 ) )
      .collect(Collectors.toList());

  System.out.println(filteredEmployees);
```

Program Output.

Output

```
[Employee [id=3, name=Charles, salary=300.0]]
```

2.3. The `or()` Method for Logical OR Operation

- The Predicate.or() method returns a composed predicate that represents a **short-circuiting logical OR of given predicate and another predicate**.
- When evaluating the composed predicate, if the first predicate is `true`, then the other predicate is not evaluated.
- Any exceptions thrown during evaluation of either predicate are relayed to the caller; if evaluation of first predicate throws an exception, the other predicate will not be evaluated.

In the given example, we are finding all the employees whose *id* is less than 2 or *salary* is greater than 500.

id is less than 2 OR salary is greater than 500

```
List<Employee> employeesList = Arrays.asList(
                new Employee(1, "Alex", 100),
                new Employee(2, "Brian", 200),
                new Employee(3, "Charles", 300),
                new Employee(4, "David", 400),
                new Employee(5, "Edward", 500),
                new Employee(6, "Frank", 600)
              );

Predicate<Employee> idLessThan2 = e -> e.getId() < 2;

Predicate<Employee> salaryGreaterThan500 = e -> e.getSalary() > 500;

List<Employee> filteredEmployees = employeesList.stream()
        .filter( idLessThan2.or( salaryGreaterThan500 ) )
        .collect(Collectors.toList());

System.out.println(filteredEmployees);
```

Program output.

Output

```
[Employee [id=1, name=Alex, salary=100.0],
Employee [id=6, name=Frank, salary=600.0]]
```

3. Conclusion

In this Java tutorial, we learned to create simple predicates and use them to filter the Stream items. Then we learned to combine multiple simple predicates to create complex predicates using *and()*, *or()* and *negate()* methods.

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. Java Predicates
2. Applying Multiple Conditions on Java Streams
3. Java Stream reuse – traverse stream multiple times?
4. Sorting a Stream by Multiple Fields in Java
5. Getting Distinct Stream Items by Comparing Multiple Fields
6. Multiple Inheritance in Java
7. Java group by sort – multiple comparators example
8. Java Enum with Multiple Values
9. Adding Multiple Items to ArrayList
0. Spring Batch – Writing to Multiple Destinations with Classifier

Leave a Comment

Name *

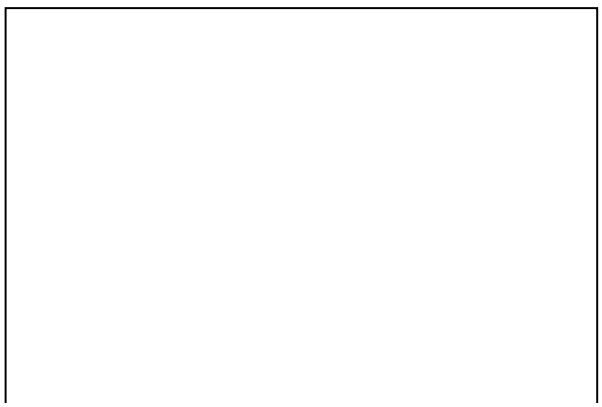
Email *

Website

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

Post Comment

Search …




VIDEOGAMER
NEW SOUTH PARK GAME
WITHOUT UBISOFT COMING?!

HowToDoInJava

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

Meta Links

- About Me
- Contact Us
- Privacy policy