**Predicate Interface in Java**

It is a functional interface which represents a predicate (boolean-valued function) of one argument. It is defined in the java.util.function package and contains test() a functional method.

|  |  |
| --- | --- |
| **Methods** | **Description** |
| boolean test(T t) | It evaluates this predicate on the given argument. |
| default Predicate<T> and(Predicate<? super T> other) | It returns a composed predicate that represents a short-circuiting logical AND of this predicate and another. When evaluating the composed predicate, if this predicate is false, then the other predicate is not evaluated. |
| default Predicate<T> negate() | It returns a predicate that represents the logical negation of this predicate. |
| default Predicate<T> or(Predicate<? super T> other) | It returns a composed predicate that represents a short-circuiting logical OR of this predicate and another. When evaluating the composed predicate, if this predicate is true, then the other predicate is not evaluated. |
| static <T> Predicate<T> isEqual(Object targetRef) | It returns a predicate that tests if two arguments are equal according to Objects.equals(Object, Object). |

Example 1:

import java.util.function.Predicate;

public class PredicateInterfaceExample {

    public static void main(String[] args) {

        Predicate<Integer> pr = a -> (a > 18); // Creating predicate

        System.out.println(pr.test(10));    // Calling Predicate method

    }

}

output:

false

**and()**

The Predicate and() method is a default method. The and() method is used to combine two other Predicate functions in the same way I showed in the beginning of this Java functional composition tutorial. Here is an example of functional composition with the Predicate and() method:

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

Predicate<String> endsWithX = (text) -> text.endsWith("x");

Predicate<String> composed = startsWithA.and(endsWithX);

String input = "A hardworking person must relax";

boolean result = composed.test(input);

System.out.println(result);

This Predicate composition example composes a new Predicate from two other Predicate instances using the and() method of one of the basic Predicate instances.

The composed Predicate will return true from it's test() method if both of the Predicate instances it was composed from also return true. In other words, if both Predicate one *and* Predicate two return true.

**or()**

The Predicate or() method is used to combine a Predicate instance with another, to compose a third Predicate instance. The composed Predicate will return true if either of the Predicate instances it is composed from return true, when their test() methods are called with same input parameter as the composed Predicate. Here is a Java Predicate or() functional composition example:

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

Predicate<String> endsWithX = (text) -> text.endsWith("x");

Predicate<String> composed = startsWithA.or(endsWithX);

String input = "A hardworking person must relax sometimes";

boolean result = composed.test(input);

System.out.println(result);

This Predicate or() functional composition example first creates two basic Predicate instances. Second, the example creates a third Predicate composed from the first two, by calling the or() method on the first Predicate and passing the second Predicate as parameter to the or() method.

The output of running the above example will be true because the first of the two Predicate instances used in the composed Predicate will return true when called with the String "A hardworking person must relax sometimes".