# Exploring Predicate Functional Interface in Java 8

## Introduction

In this tutorial, we will explore the \*\*Predicate Functional Interface\*\* introduced in Java 8. We will examine its abstract methods, default methods, and various operations including \*\*negate(), and(), or()\*\*, using real-world coding examples.

## Overview of Predicate Functional Interface

1. The \*\*Predicate Functional Interface\*\* is available in the `java.util.function` package.

2. It contains a single abstract method:

- \*\*Boolean test(T t)\*\*: Accepts an input, performs an operation, and returns a boolean result.

3. It also supports additional \*\*default methods\*\*:

- \*\*and()\*\*: Combines multiple predicates using logical AND.

- \*\*or()\*\*: Combines multiple predicates using logical OR.

- \*\*negate()\*\*: Reverses the boolean result of a predicate.

## Implementing Predicate Functional Interface

1. Create a class \*\*PredicateExample\*\*.

2. Implement a predicate to check whether a given integer is even.

### \*\*Code Implementation:\*\*

import java.util.function.Predicate;  
  
public class PredicateExample {  
 public static void main(String[] args) {  
 Predicate<Integer> p = (n) -> n % 2 == 0;  
 System.out.println("Is 4 even? " + p.test(4));  
 }  
}

## Optimizing Predicate Implementation

1. If the Lambda body contains a \*\*single statement\*\*, we can remove curly braces `{}` and the `return` keyword.

2. Optimized implementation:

### \*\*Code Implementation:\*\*

Predicate<Integer> p1 = n -> n % 2 == 0;  
System.out.println("Is 4 even? " + p1.test(4));

## Using Predicate with Logical Operators

1. \*\*Combining Predicates Using and()\*\*: Check if a number is divisible by both \*\*2 and 5\*\*.

2. \*\*Using or()\*\*: Check if a number is divisible by either \*\*2 or 5\*\*.

3. \*\*Using negate()\*\*: Reverse the predicate result.

### \*\*Code Implementation:\*\*

Predicate<Integer> p2 = (n) -> n % 5 == 0;  
  
// Using and() to combine predicates  
System.out.println("Is 10 divisible by 2 and 5? " + p1.and(p2).test(10));  
  
// Using or() to combine predicates  
System.out.println("Is 9 divisible by 2 or 5? " + p1.or(p2).test(9));  
  
// Using negate() to reverse result  
System.out.println("Negated result for 10: " + p1.and(p2).negate().test(10));

## Predicate Chaining and Code Reusability

1. \*\*Predicate chaining\*\* allows us to combine multiple conditions without duplicating code.

2. \*\*Reusability\*\*: The same predicate logic can be reused across multiple sections of an application.

3. Example: Checking if a number is even and greater than 10.

### \*\*Code Implementation:\*\*

Predicate<Integer> p3 = (n) -> n > 10;  
  
// Chaining predicates  
System.out.println("Is 12 even and greater than 10? " + p1.and(p3).test(12));

## Conclusion

- Java 8 introduced \*\*Predicate Functional Interface\*\* to simplify boolean condition checks.

- The \*\*test()\*\* method is used to evaluate a condition.

- The \*\*and(), or(), and negate()\*\* methods help in predicate composition.

- Predicate chaining allows for \*\*code reusability and efficient condition checking\*\*.

- In the next tutorial, we will apply Predicate Functional Interface in \*\*real-world scenarios using student data\*\*.

### \*\*End of Tutorial\*\*

Thank you for watching!