Java Predicate Functional Interface Documentation

The java.util.function.Predicate<T> interface is a key part of Java's functional programming features introduced in **Java 8**. It's designed to represent a **boolean-valued function** (a predicate) of one argument.

1. Overview and Definition

Aspect	Description
Package	java.util.function
Type	**Functional Interface**
Purpose	To define a single-argument function that returns a **boolean**. Used primarily for **filtering** collections and data structures.
SAM Method	The Single Abstract Method (SAM) is test(T t).

A Predicate<T> takes an object of type T and determines if it satisfies some condition, often acting as a test.

Definition:

Java

```
@FunctionalInterface
public interface Predicate<T> {
    /**
     * Evaluates this predicate on the given argument.
     *
     * @param t the input argument
     * @return true if the input argument matches the predicate, otherwise false
     */
    boolean test(T t);

    // ... default and static methods for composition
}
```

---

2. Usage and Examples

The Predicate interface is typically implemented using **Lambda Expressions** or **Method References**.

2.1. Basic Implementation (Lambda Expression)

Example: A predicate to check if an integer is even.

Java

```
import java.util.function.Predicate;

public class PredicateExample {
    public static void main(String[] args) {
        // Predicate<Integer> implemented using a lambda
        Predicate<Integer> isEven = (number) -> number % 2 == 0;
```

```
        // Testing the predicate
        System.out.println("Is 4 even? " + isEven.test(4));
        System.out.println("Is 7 even? " + isEven.test(7));
    }
}
```

2.2. Filtering Collections (Streams API)

The most common use case is with the **Java Streams API**'s filter() method, which accepts a Predicate.

Example: Filtering a list of strings to only include those with a length greater than 5.

Java

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class StreamFilterExample {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("apple", "banana", "cat", "dogma", "elephant");

        // Predicate to check length > 5
        Predicate<String> isLong = s -> s.length() > 5;

        // Use the predicate in the stream's filter method
        List<String> longWords = words.stream()
                                      .filter(isLong)
                                      .collect(Collectors.toList());

        System.out.println(longWords); // Output: [banana, elephant]
    }
}
```

---

3. Default and Static Methods

The Predicate interface provides powerful methods for combining and negating logic, making complex filtering reusable and readable.

Method	Type	Description
and(Predicate<? super T> other)	Default	Represents a **logical AND**. Evaluates the other predicate only if this one is true.
or(Predicate<? super T> other)	Default	Represents a **logical OR**. Evaluates the other predicate only if this one is false.
negate()	Default	Returns a predicate that represents the **logical NOT** of this predicate.
isEqual(Object targetRef)	Static	Returns a predicate that tests if an argument is **equal** to the targetRef.
not(Predicate<? super T> target)	Static (Java 11+)	Returns the **negation** of the supplied predicate.

3.1. Combining Predicates (and, or, negate)

Example: Combining and negating integer predicates.

Java

```
import java.util.function.Predicate;

public class PredicateComposition {
    public static void main(String[] args) {
        // Base Predicates
        Predicate<Integer> isGreaterThanTen = x -> x > 10;
        Predicate<Integer> isLessThanTwenty = x -> x < 20;

        // AND composition: 10 < x < 20
        Predicate<Integer> isInRange = isGreaterThanTen.and(isLessThanTwenty);
        System.out.println("15 in range? " + isInRange.test(15)); // true

        // NEGATE: NOT (x > 10) -> (x <= 10)
        Predicate<Integer> isNotGreaterThanTen = isGreaterThanTen.negate();
        System.out.println("10 is not > 10? " + isNotGreaterThanTen.test(10)); // true
    }
}
```

---

4. Related Primitive Interfaces

Java provides specialized versions of Predicate for primitive types to avoid the performance overhead of **autoboxing** and **unboxing**.

Interface	Primitive Type	SAM Method	Purpose
IntPredicate	int	boolean test(int value)	For int primitive type.
LongPredicate	long	boolean test(long value)	For long primitive type.
DoublePredicate	double	boolean test(double value)	For double primitive type.

BiPredicate<T, U>	Generic T, U	boolean test(T t, U u)	A predicate that takes **two arguments**.

Example (IntPredicate):

Java

```
import java.util.function.IntPredicate;

public class IntPredicateExample {
    public static void main(String[] args) {
        // IntPredicate for int primitive
        IntPredicate isNegative = i -> i < 0;

        System.out.println("Is -5 negative? " + isNegative.test(-5)); // true
    }
}
```