# Function interface- A functional interface in the java.util.function package in Java 8

# <https://www.quora.com/What-is-predicate-function-and-consumer-in-Java-1-8>

<https://www.javabrahman.com/java-8/java-8-java-util-function-consumer-tutorial-with-examples/>

Java8 landed with new feature so called functional Interface.These predicate, function and consumer are nothing but builtin Functional Interfaces which are part of java.util.function Package.

So here, What's functional Interface ? This concept says the Interface which can have only single abstract Method and lambda expressions can be used to represent the instance of a functional interface. Also, We can have any number of Default methods in any functional interface.

Functional interfaces like Runnable, Clonnable, Comparable are also some of the best examples of functional interfaces.

1. **Predicate Functional Interface**

This interface has an abstract method test(T t) which has Boolean return type.Usage, when we need to return/check the condition as True or False. It is best suited to code.

2. **Function Functional Interface**

This interface has an abstract method apply which takes argument of type T and returns a result of type R. Here, R is nothing but the type of result user wants to return. It may be Integer, String, Boolean , Double, Long.

Function interface has two methods:

[**R apply(T t)**](http://download.java.net/lambda/b82/docs/api/java/util/function/Function.html#apply(T)) – Compute the result of applying the function to the input argument

[**default ‹V› Function‹T,V›**](http://download.java.net/lambda/b82/docs/api/java/util/function/Function.html#compose(java.util.function.Function)) – Combine with another function returning a function which performs both functions.

Function interface is uses in cases where you want to encapsulate some code into a method which accepts some value as an input parameter and then returns another value after performing required operations on the input. The input parameter type and the return type of the method can either be same or different.

# Java 8 Predicate with Examples

A [Functional Interface](https://www.geeksforgeeks.org/functional-interfaces-java/) is an Interface which allows only one Abstract method within the Interface scope. There are some predefined functional interface in Java like Predicate, consumer, supplier etc. The return type of a Lambda function (introduced in JDK 1.8) is a also functional interface.

The Functional Interface **PREDICATE** is defined in the *java.util.Function package*. It improves manageability of code, helps in unit-testing them separately, and contain some methods like:

1. **isEqual(Object targetRef) :**Returns a predicate that tests if two arguments are equal according to Objects.equals(Object, Object).
2. **static Predicate isEqual(Object targetRef)**
3. Returns a predicate that tests if two arguments are
4. equal according to Objects.equals(Object, Object).
5. **T :** the type of arguments to the predicate
6. **Parameters:**
7. **targetRef :** the object reference with which to
8. compare for equality, which may be null
9. **Returns:** a predicate that tests if two arguments
10. are equal according to Objects.equals(Object, Object)
11. **and(Predicate other) :**Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
12. **default Predicate and(Predicate other)**
13. Returns a composed predicate that represents a
14. short-circuiting logical AND of this predicate and another.
15. **Parameters:**
16. other: a predicate that will be logically-ANDed with this predicate
17. Returns : a composed predicate that represents the short-circuiting
18. logical AND of this predicate and the other predicate

Throws: NullPointerException - if other is null

1. **negate() :** Returns a predicate that represents the logical negation of this predicate.
2. **default Predicate negate()**
3. Returns:a predicate that represents the logical

negation of this predicate

1. **or(Predicate other) :** Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
2. **default Predicate or(Predicate other)**
3. Parameters:
4. other : a predicate that will be logically-ORed with this predicate
5. Returns:
6. a composed predicate that represents the short-circuiting
7. logical OR of this predicate and the other predicate

Throws : NullPointerException - if other is null

1. **test(T t) :** Evaluates this predicate on the given argument.boolean test(T t)
2. **test(T t)**
3. Parameters:
4. t - the input argument
5. Returns:

true if the input argument matches the predicate, otherwise false

|  |
| --- |
| import java.util.function.Predicate;  public class PredicateInterfaceExample1 {      public static void main(String[] args)      {          // Creating predicate          Predicate<Integer> lesserthan = i -> (i < 18);            // Calling Predicate method          System.out.println(lesserthan.test(10));      }  } |

Output:

Supplier Interface in Java with Examples

The **Supplier Interface** is a part of the **java.util.function** package which has been introduced since Java 8, to implement [functional programming](https://www.geeksforgeeks.org/functional-programming-paradigm/) in Java. It represents a function which does not take in any argument but produces a value of type T.

Hence this functional interface takes in only one generic namely:-

* **T**: denotes the type of the result

The lambda expression assigned to an object of Supplier type is used to define its **get()** which eventually produces a value. Suppliers are useful when we don’t need to supply any value and obtain a result at the same time.

The Supplier interface consists of only one function:

### 1. get()

This method does not take in any argument but produces a value of type T.

**Syntax:**

T get()

**Returns:** This method returns a **value** of type T.

Below is the code to illustrate get() method:

**Program:**

|  |
| --- |
| import java.util.function.Supplier;    public class Main {      public static void main(String args[])      {            // This function returns a random value.          Supplier<Double> randomValue = () -> Math.random();            // Print the random value using get()          System.out.println(randomValue.get());      }  } |