***Functional interfaces***

***Functional interfaces* provide target types for lambda expressions and method references.**

A *functional interface* in Java is an interface that contains only a single abstract method.

***Important points of Functional interfaces: -***

* Only One abstract method.
* Can contain **any number of default method**, which do have an implementation.
* Can contain **any number of static method**, which do have an implementation.
* Can contain **any number of** **default** **method** and **any number of** **static method**, which do have an implementation.
* Can be implemented by a Java Lambda Expression.
* Java contains a set of functional interfaces designed for commonly occuring use cases, so you don't have to create your own functional interfaces for every little use case.
* It's recommended that all functional interfaces have an informative **@FunctionalInterface annotation**. This clearly communicates the purpose of the interface, and also allows a compiler to generate an error if the annotated interface does not satisfy the conditions.
* Functional interfaces are organised under **java.util.function package**.

**@FunctionalInterface Annotation**

**@FunctionalInterface** annotation is introduced in Java 8 to represent functional interfaces. Although, **it is not compulsory** to write functional interface using this annotation.

But, if you are using **@FunctionalInterface** annotation then your interface should contain only one abstract method. If you try to write more than one abstract method, compiler will show the error.

**java.util.function package**

All Java 8 functional interfaces are organised in java.util.function package. Each functional interface in this package represents an operation that can be performed by the lambda expression.

**Some of Important Functional Interface are followings:-**

**Predicate Functional Interface: -**

This is a functional interface whose functional method is **test(Object**). It can be use as the assignment target for a lambda expression or method reference.

**Whe to use Predicate Interface?**

**Answer: -** Use this Interface when you want to evaluate a boolean expresson which takes an argument of type T.

**For example: -** String::isEmpty, Character::isDigit

**Methods of Predicate Interface are followings:-**

* **default Predicate<T>and(Predicate<? super T> other):** Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
* **static <T> Predicate<T>isEqual(Object targetRef):** Returns a predicate that tests if two arguments are equal according to Objects.equals(Object, Object).
* **default Predicate<T>negate():** Returns a predicate that represents the logical negation of this predicate.
* **default Predicate<T>or(Predicate<? super T> other):** Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
* **boolean test(T t):** Evaluates this predicate on the given argument.

**Related Functional Interfaces to support Primitive Types:-**

* IntPredicate
* LongPredicate
* DoublePredicate

**Consumer Functional Interface: -**

The Java Consumer interface is a functional interface that represents an function that consumes a value without returning any value. A Java Consumer implementation could be printing out a value, or writing it to a file, or over the network etc.

**Whe to use Consumer Interface?**

**Answer: -** Use Consumer Interface when you want to perfomrm some operations on an Object, like printing out a value, or writing it to a file etc.

**For example: -** System.out::println, Error::printStackTrace

**Methods of Consumer Interface are followings:-**

* **void accept(T t):-** Performs this operation on the given argument.
* **default Consumer<T>andThen(Consumer<? super T> after):-**Returns a composed Consumer that performs, in sequence, this operation followed by the after operation.

**Note: - accept(T t) is a functional method.**

**Related Functional Interfaces to support Primitive Types:-**

* IntConsumer Interface.
* LongConsumer Interface.
* DoubleConsumer Interface

**Supplier Functional Interface: -**

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

**Supplier interface takes no arguments and returns a result.**

**Whe to use Supplier Interface?**

**Answer: -** Use this interface when you want to create new Objects.

**For example: -** LocalDate::now, Instant::now

**Methods of Supplier Interface are followings:-**

**T get():-** Gets a result.

**Related Functional Interfaces to support Primitive Types:-**

* BooleanSupplier Interface.
* IntSupplier Interface.
* LongSupplier Interface.
* DoubleSupplier Interface.

**Function Functional Interface: -**

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

Represents a function **that accepts one argument and produces a result.** This is a functional interface whose functional method is **apply(Object).**

**Whe to use Function Functional Interface?**

**Answer: -** Use this interface when you want to extract a data from and existing data.

**For example: -** Arrays::asList, Integer::toBinaryString

**Methods of Function Interface are followings:-**

* **default <V> Function<T, V> andThen (Function<? super R, ? extends V> after): -** Returns a composed function that first applies this function to its input, and then applies the after function to the result.
* **R apply (T t): -** Applies this function to the given argument.
* **default <V> Function<V, R> compose (Function<? super V, ? extends T> before): -** Returns a composed function that first applies the before function to its input, and then applies this function to the result.
* **static <T> Function<T, T> identity(): -** Returns a function that always returns its input argument.

**Related Functional Interfaces to support Primitive Types:-**

* IntFunction Interface
* LongFunction Interface
* DoubleFunction Interface
* ToIntFunction Interface
* ToLongFunction Interface
* ToDobleFunction Interface
* IntToLongFunction Interface
* IntToDoubleFunction Interface
* LongToDoubleFunction Interface
* LongToIntFunction Interface
* DoubleToIntFunction Interface
* DoubleToLongFunction Interface

**BiPredicate Functional Interface: -**

BiPredicate is a functional interface, which accepts two arguments and returns a boolean, basically this BiPredicate is same with the Predicate, instead, it takes 2 arguments for the test.

It’s functional method is **test(T t, U u).**

**Whe to use BiPredicate Functional Interface?**

**Answer: -** Use this interface when you want to evaluate a boolean expression of two arguments.

**Methods of BiPredicate Interface are followings:-**

* **default BiPredicate<T, U> and (BiPredicate<? super T, ? super U> other): -** Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
* **default BiPredicate<T, U> negate(): -** Returns a predicate that represents the logical negation of this predicate.
* **default BiPredicate<T, U> or (BiPredicate<? super T, ? super U> other): -** Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
* **boolean test (T t, U u): -** Evaluates this predicate on the given arguments.

**BiConsumer Functional Interface: -**

BiConsumer is a functional interface; it takes two arguments and returns nothing.

It’s functional method is **accept(T t, U u).**

**Whe to use BiConsumer Functional Interface?**

**Answer: -** Use this interface when you want to perform some operation on two objects.

**Methods of BiConsumer Interface are followings:-**

* **void accept (T t, U u): -**Performs this operation on the given arguments.
* **default BiConsumer<T, U> andThen (BiConsumer<? super T, ? super U> after): -** Returns a composed BiConsumer that performs, in sequence, this operation followed by the after operation.

**Related Functional Interfaces to support Primitive Types:-**

* ObjIntConsumer Interface.
* ObjLongConsumer Interface.
* ObjDoubleConsumer Interface.

**BiFunction Functional Interface: -**

BiFunction is a functional interface; it takes two arguments and returns an object.

It’s functional method is **apply(Object, Object)**.

**Whe to use BiFunction Functional Interface?**

**Answer: -** Use this interface when you want to extract result data from two existing objects.

**Methods of BiFunction Interface are followings:-**

* **default <V> BiFunction<T, U, V> andThen (Function<? super R, ? extends V> after): -** Returns a composed function that first applies this function to its input, and then applies the after function to the result.
* **R apply (T t, U u): -** Applies this function to the given arguments.

**Related Functional Interfaces to support Primitive Types:-**

* ToIntBiFunction Interface
* ToLongBiFunction Interface
* ToDoubleBiFunction Interface

**UnaryOperator Functional Interface: -**

UnaryOperator is a functional interface and it extends Function. The UnaryOperator takes one argument, and returns a result of the same type of its arguments.

It’s functional method is **Function.apply(Object)**.

**Whe to use UnaryOperator Functional Interface?**

**Answer: -** Use this interface when you want to extract a data from and existing data. For example: - String::toLowerCase, Math::tan

**Methods of UnaryOperator Interface are followings:-**

1. **static <T> UnaryOperator<T> identity():-** Returns a unary operator that always returns its input argument.

**Related Functional Interfaces to support Primitive Types:-**

* IntUnaryOperator Interface.
* LongUnaryOperator Interface.
* DoubleUnaryOperator Interface.

**BinaryOperator Functional Interface: -**

BinaryOperator is a functional interface and it extends BiFunction. The BinaryOperator takes two arguments of the same type and returns a result of the same type of its arguments.

It's functional method is **BiFunction.apply(Object, Object).**

**Whe to use BinaryOperator Functional Interface?**

**Answer: -** Use this interface when you want to extract result data from two existing objects. For example:- BigInteger::add, Math::pow

**Methods of BinaryOperator Interface are followings:-**

1. **static <T> BinaryOperator<T> maxBy (Comparator<? super T> comparator): -** Returns a BinaryOperator which returns the greater of two elements according to the specified Comparator.
2. **static <T> BinaryOperator<T> minBy (Comparator<? super T> comparator): -** Returns a BinaryOperator which returns the lesser of two elements according to the specified Comparator.

**Related Functional Interfaces to support Primitive Types:-**

* IntBinaryOperator Interface.
* LongBinaryOperator Interface.
* DoubleBinaryOperator Interface.

**-------------------------------------------------------------------------------------------------------**

**Module** [java.base](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/module-summary.html)

**Since: 1.8**

# Package java.util.function

package java.util.function

*Functional interfaces* provide target types for lambda expressions and method references. Each functional interface has a single abstract method, called the *functional method* for that functional interface, to which the lambda expression's parameter and return types are matched or adapted. Functional interfaces can provide a target type in multiple contexts, such as assignment context, method invocation, or cast context

**Interface Summary**

[**BiConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BiConsumer.html)<T, U>

Represents an operation that accepts two input arguments and returns no result.

[**BiFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BiFunction.html)<T, U, R>

Represents a function that accepts two arguments and produces a result.

[**BinaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BinaryOperator.html)<T>

Represents an operation upon two operands of the same type, producing a result of the same type as the operands.

[**BiPredicate**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BiPredicate.html)<T, U>

Represents a predicate (boolean-valued function) of two arguments.

[**BooleanSupplier**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/BooleanSupplier.html)

Represents a supplier of boolean-valued results.

[**Consumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Consumer.html)<T>

Represents an operation that accepts a single input argument and returns no result.

[**DoubleBinaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleBinaryOperator.html)

Represents an operation upon two double-valued operands and producing a double-valued result.

[**DoubleConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleConsumer.html)

Represents an operation that accepts a single double-valued argument and returns no result.

[**DoubleFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleFunction.html)<R>

Represents a function that accepts a double-valued argument and produces a result.

[**DoublePredicate**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoublePredicate.html)

Represents a predicate (boolean-valued function) of one double-valued argument.

[**DoubleSupplier**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleSupplier.html)

Represents a supplier of double-valued results.

[**DoubleToIntFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleToIntFunction.html)

Represents a function that accepts a double-valued argument and produces an int-valued result.

[**DoubleToLongFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleToLongFunction.html)

Represents a function that accepts a double-valued argument and produces a long-valued result.

[**DoubleUnaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/DoubleUnaryOperator.html)

Represents an operation on a single double-valued operand that produces a double-valued result.

[**Function**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Function.html)<T, R>

Represents a function that accepts one argument and produces a result.

[**IntBinaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntBinaryOperator.html)

Represents an operation upon two int-valued operands and producing an int-valued result.

[**IntConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntConsumer.html)

Represents an operation that accepts a single int-valued argument and returns no result.

[**IntFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntFunction.html)<R>

Represents a function that accepts an int-valued argument and produces a result.

[**IntPredicate**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntPredicate.html)

Represents a predicate (boolean-valued function) of one int-valued argument.

[**IntSupplier**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntSupplier.html)

Represents a supplier of int-valued results.

[**IntToDoubleFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntToDoubleFunction.html)

Represents a function that accepts an int-valued argument and produces a double-valued result.

[**IntToLongFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntToLongFunction.html)

Represents a function that accepts an int-valued argument and produces a long-valued result.

[**IntUnaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/IntUnaryOperator.html)

Represents an operation on a single int-valued operand that produces an int-valued result.

[**LongBinaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongBinaryOperator.html)

Represents an operation upon two long-valued operands and producing a long-valued result.

[**LongConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongConsumer.html)

Represents an operation that accepts a single long-valued argument and returns no result.

[**LongFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongFunction.html)<R>

Represents a function that accepts a long-valued argument and produces a result.

[**LongPredicate**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongPredicate.html)

Represents a predicate (boolean-valued function) of one long-valued argument.

[**LongSupplier**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongSupplier.html)

Represents a supplier of long-valued results.

[**LongToDoubleFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongToDoubleFunction.html)

Represents a function that accepts a long-valued argument and produces a double-valued result.

[**LongToIntFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongToIntFunction.html)

Represents a function that accepts a long-valued argument and produces an int-valued result.

[**LongUnaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/LongUnaryOperator.html)

Represents an operation on a single long-valued operand that produces a long-valued result.

[**ObjDoubleConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ObjDoubleConsumer.html)<T>

Represents an operation that accepts an object-valued and a double-valued argument, and returns no result.

[**ObjIntConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ObjIntConsumer.html)<T>

Represents an operation that accepts an object-valued and a int-valued argument, and returns no result.

[**ObjLongConsumer**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ObjLongConsumer.html)<T>

Represents an operation that accepts an object-valued and a long-valued argument, and returns no result.

[**Predicate**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Predicate.html)<T>

Represents a predicate (boolean-valued function) of one argument.

[**Supplier**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/Supplier.html)<T>

Represents a supplier of results.

[**ToDoubleBiFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ToDoubleBiFunction.html)<T, U>

Represents a function that accepts two arguments and produces a double-valued result.

[**ToDoubleFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ToDoubleFunction.html)<T>

Represents a function that produces a double-valued result.

[**ToIntBiFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ToIntBiFunction.html)<T, U>

Represents a function that accepts two arguments and produces an int-valued result.

[**ToIntFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ToIntFunction.html)<T>

Represents a function that produces an int-valued result.

[**ToLongBiFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ToLongBiFunction.html)<T, U>

Represents a function that accepts two arguments and produces a long-valued result.

[**ToLongFunction**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/ToLongFunction.html)<T>

Represents a function that produces a long-valued result.

[**UnaryOperator**](https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/function/UnaryOperator.html)<T>

Represents an operation on a single operand that produces a result of the same type as its operand.