

Function Interface

This lesson explains the Function interface in Java.

We'll cover the following

- Function
 - R apply(T t)
 - compose(Function before)
 - andThen(Function after)
- BiFunction

Function is a category of functional interfaces that takes an object of type **T** and returns an object of type **R**.

Until now, the functional interfaces that we've discussed have either not taken any argument(**Supplier**), not returned any value(**Consumer**), or returned only a boolean(**Predicate**).

Function interfaces are very useful as we can specify the type of input and output.

Below are some of the interfaces that fall in this category.

Interface	Purpose	Abstract Method
Function<T,R>	Represents a function that accepts one argument and produces a result	R apply (T t)
DoubleFunction<R>	Accepts a double-valued argument and produces a result	R apply (double t)
IntFunction<R>	Accepts an int-valued argument and produces a result	R apply (int t)
LongFunction<R>	Accepts a long-valued argument and produces a result	R apply (long t)
DoubleToIntFunction	Accepts a double-valued argument and produces an int-valued result	int applyAsInt(double value)
DoubleToLongFunction	Accepts a double-valued argument and produces a long-valued result	long applyAsLong(double value)
IntToDoubleFunction	Accepts an int-valued argument and produces a double-valued result	double applyAsDouble(int value)
IntToLongFunction	Accepts an int-valued argument and produces a long-valued result	long applyAsLong(int value)
LongToIntFunction	Accepts a long-valued argument and produces an int-valued result	int applyAsInt(long value)
LongToDoubleFunction	Accepts a long-valued argument and produces a double-valued result.	double applyAsDouble(long value)
ToDoubleFunction<T>	Accepts a reference type and produces an int-valued result	double applyAsDouble(T value)
ToIntFunction<T>	Accepts a reference type and produces an int-valued result	int applyAsInt(T value)
ToLongFunction<T>	Accepts a reference type and produces a long-valued result.	long applyAsLong(T value)
BiFunction<T,U,R>	Represents a function that accepts two arguments and produces a result	R apply(T t, U u)
ToDoubleBiFunction<T,U>	Accepts two reference type arguments and produces a double-valued result	R applyAsDouble(T t, U u)
ToIntBiFunction<T,U>	Accepts two reference type arguments and produces an int-valued result	R applyAsInt(T t, U u)
ToLongBiFunction<T,U>	Accepts two reference type arguments and produces a long-valued result	R applyAsLong(T t, U u)

Let us discuss some types of `Function` functional interfaces.

`Function<T, R>`

The function takes only one argument of type `T` and returns a result of type `R`.

The following is the list of all the methods in the `Function<T, R>` interface.

Method Summary

All Methods	Static Methods	Instance Methods	Abstract Methods	Default Methods
Modifier and Type		Method and Description		
default	<V> <code>Function<T,V></code>	<code>andThen(Function<? super R,? extends V> after)</code> Returns a composed function that first applies this function to its input, and then applies the after function to the result.		
	<code>R</code>	<code>apply(T t)</code> Applies this function to the given argument.		
default	<V> <code>Function<V,R></code>	<code>compose(Function<? super V,? extends T> before)</code> Returns a composed function that first applies the before function to its input, and then applies this function to the result.		
static	<T> <code>Function<T,T></code>	<code>identity()</code> Returns a function that always returns its input argument.		

Let's look at an example of each method:

`R apply(T t)`

This is the abstract method of the `Function` interface. It takes one argument of type `T` as input and returns a value of type `R`.

In the below example, we will create a function called `lengthFunction`. It takes a string as input and returns the length of the string as output.

```
import java.util.function.Function;

public class FunctionInterfaceDemo {

    public static void main(String[] args) {
        // Created a function which returns the length of string.
        Function<String, Integer> lengthFunction = str -> str.length();

        System.out.println("String length: " + lengthFunction.apply("This is awesome!!"));
    }
}
```

`compose(Function<? super V, ? extends T> before)`

Returns a composed function that first applies the function provided as a parameter on the input, and then applies the function on which it is called, to the

parameter on the input, and then applies the function on which it is called, to the result.

```
import java.util.function.Function;

public class FunctionDemo {

    public static void main(String args[]) {

        // Function which adds 10 to the given element.
        Function<Integer, Integer> increment = x -> x + 10;
        // Function which doubles the given element.
        Function<Integer, Integer> multiply = y -> y * 2;
        // Since we are using compose(), multiplication will be done first and then increment will
        System.out.println("compose result: " + increment.compose(multiply).apply(3));

    }
}
```



andThen(Function<? super R,? extends V> after) #

This method returns a composed function that first applies the function on which it is called on the input, and then applies the function provided as parameter, to the result.

```
import java.util.function.Function;

public class FunctionDemo {

    public static void main(String args[]) {
        Function<Integer,Integer> increment = x -> x + 10;
        Function<Integer,Integer> multiply = y -> y * 2;
        // Since we are using andThen(), increment will be done first and then multiplication will
        System.out.println("andThen result: " + increment.andThen(multiply).apply(3));

    }
}
```



BiFunction<T,U,R> #

The **BiFunction<T, U, R>** is similar to **Function<T, R>** interface; the only difference is that the **BiFunction** interface takes in two parameters and returns an output.

Below is the list of methods in the **BiFunction** interface.

Method Summary

All Methods	Instance Methods	Abstract Methods	Default Methods
Modifier and Type		Method and Description	
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.	

In the below example, we will create a **BiFunction** that takes two numbers as input and returns their sum.

```
import java.util.function.BiFunction;

public class BiFunctionInterfaceDemo {
    public static void main(String args[])
    {

        BiFunction<Integer, Integer, Integer> add = (a, b) -> a + b;

        System.out.println("Sum = " + add.apply(2, 3));

    }
}
```

In the next lesson, you will learn about a subtype of the **Function** interface, the **Unary** operator.