**Functional Interface in Java**

The **Function Interface** is a part of the **java.util.function** package which has been introduced since Java 8, to implement functional programming in Java. It represents a function that takes in one argument and produces a result. Hence this functional interface takes 2 generics namely as follows:

* **T**: denotes the type of the input argument
* **R**: denotes the return type of the function

Note: *The lambda expression assigned to an object of Function type is used to define its****apply()****which eventually applies the given function on the argument.*

**Methods in Functional Interface**

The Function interface consists of the following 4 methods as listed which are later discussed as follows:

1. apply()
2. compose()
3. andThen()
4. identity()

**Method 1: apply()**

Syntax:

R apply(T t)

**Parameters:** This method takes in only one parameter **t** which is the function argument.

**Return Type:**This method returns the **function result** which is of type R.

Example:

// Java Program to Illustrate Functional Interface

// Via apply() method

// Importing interface

import java.util.function.Function;

// Main class

public class GFG {

    // Main driver method

    public static void main(String args[])

    {

        // Function which takes in a number

        // and returns half of it

        Function<Integer, Double> half = a -> a / 2.0;

        // Applying the function to get the result

        System.out.println(half.apply(10));

    }

}

Output : 5.0

**Method 2: compose()**

The Java Function **compose**() method composes a new Function instance from the Function instance it is called on, and the Function instance passed as parameter to the compose() method.

The Function returned by compose() will first call the Function passed as a parameter to compose(), and then it will call the Function which compose() was called on. This is easier to understand with an example, so here is a Java Function compose() example:

Function<Integer, Integer> multiply = (value) -> value \* 2;

Function<Integer, Integer> add = (value) -> value + 3;

Function<Integer, Integer> addThenMultiply = multiply.compose(add);

Integer result1 = addThenMultiply.apply(3);

System.out.println(result1);

When called with the value 3, the composed Function will first call the add Function and then the multiply Function. The resulting calculation will be (3 + 3) \* 2 and the result will be 12.

Example:

// Java Program to illustrate compose() method

// Importing Function interface

import java.util.function.Function;

// Main class

public class GFG {

    // Main driver method

    public static void main(String args[])

    {

        // Function which takes in a number and

        // returns half of it

        Function<Integer, Double> half = a -> a / 2.0; (2nd Executed Method)

        // However treble the value given to half function

        half = half.compose(a -> 3 \* a); (1st Executed method)

        // Applying the function to get the result

        System.out.println(half.apply(5));

    }

}

Output: 7.5

**Method 3: andThen()**

The Java Function **andThen**() method works opposite of the compose() method. A Function composed with **andThen**() will first call the Function that **andThen**() was called on, and then it will call the Function passed as parameter to the andThen() method. Here is a Java Function andThen() example:

Function<Integer, Integer> multiply = (value) -> value \* 2;

Function<Integer, Integer> add = (value) -> value + 3;

Function<Integer, Integer> multiplyThenAdd = multiply.andThen(add);

Integer result2 = multiplyThenAdd.apply(3);

System.out.println(result2);

This example first creates a multiply Function and an add Function. Then the andThen() method is called on the multiply Function to compose a new Function, passing the add Function as parameter to andThen().

Calling the Function composed by andThen() with the value 3 will result in the following calculation  
3 \* 2 + 3 and the result will be 9.

Example:

// Java Program to illustrate addThen() method

// Importing interface

import java.util.function.Function;

// Main class

public class GFG {

    // main driver method

    public static void main(String args[])

    {

        // Function which takes in a number and

        // returns half of it

        Function<Integer, Double> half = a -> a / 2.0;

        // Now treble the output of half function

        half = half.andThen(a -> 3 \* a);

        // Applying the function to get the result

        // and printing on console

        System.out.println(half.apply(10));

    }

}

Output:

15.0

Note: As mentioned in the beginning, andThen() works opposite of compose(). Therefore, calling a.**andThen**(b) is actually the same as calling b.**compose**(a) .

**Method 4: identity()**

This method returns a function that returns its only argument.

Syntax:

static <T> Function<T, T> identity()

where T denotes the type of the argument and the value to be returned

Returns: This method returns a function that returns its own argument

**Example**:

|  |
| --- |
| // Java Program to Illustrate identity() method    // Importing Function interface  import java.util.function.Function;    // Main class  public class GFG {        // Main driver method      public static void main(String args[])      {          // Function which takes in a number and          // returns it          Function<Integer, Integer> i = Function.identity();            // Print statement          System.out.println(i.apply(10));      }  } |

// Java Program to Illustrate identity() method

// Importing Function interface

import java.util.function.Function;

// Main class

public class GFG {

    // Main driver method

    public static void main(String args[])

    {

        // Function which takes in a number and

        // returns it

        Function<Integer, Integer> i = Function.identity();

        // Print statement

        System.out.println(i.apply(10));

    }

}

Output:

10