JAVA 8 - FUNCTIONAL INTERFACES

Functional interfaces have a single functionality to exhibit. For example, a Comparable interface with a single method 'compareTo' is used for comparison purpose. Java 8 has defined a lot of functional interfaces to be used extensively in lambda expressions. Following is the list of functional interfaces defined in java.util.Function package.

S. No.	Interface & Description
1	**BiConsumer<T,U>** Represents an operation that accepts two input arguments, and returns no result.
2	**BiFunction<T,U,R>** Represents a function that accepts two arguments and produces a result.
3	**BinaryOperator<T>** Represents an operation upon two operands of the same type, producing a result of the same type as the operands.
4	**BiPredicate<T,U>** Represents a predicate $Boolean-valuedfunction$ of two arguments.
5	**BooleanSupplier** Represents a supplier of Boolean-valued results.
6	**Consumer<T>** Represents an operation that accepts a single input argument and returns no result.
7	**DoubleBinaryOperator** Represents an operation upon two double-valued operands and producing a double-valued result.
8	**DoubleConsumer** Represents an operation that accepts a single double-valued argument and returns no result.
9	**DoubleFunction<R>** Represents a function that accepts a double-valued argument and produces a result.
10	**DoublePredicate** Represents a predicate $Boolean-valuedfunction$ of one double-valued argument.
11	**DoubleSupplier**

Represents a supplier of double-valued results.

12 **DoubleToIntFunction**

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

13 **DoubleToLongFunction**

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

14 **DoubleUnaryOperator**

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

15 **Function<T,R>**

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

16 **IntBinaryOperator**

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

17 **IntConsumer**

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

18 **IntFunction<R>**

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

19 **IntPredicate**

Represents a predicate $Boolean-valuedfunction$ of one int-valued argument.

20 **IntSupplier**

Represents a supplier of int-valued results.

21 **IntToDoubleFunction**

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

22 **IntToLongFunction**

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

23 **IntUnaryOperator**

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

24 **LongBinaryOperator**

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

25 **LongConsumer**

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

26 **LongFunction<R>**

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

27 **LongPredicate**

Represents a predicate $Boolean-valuedfunction$ of one long-valued argument.

28 **LongSupplier**

Represents a supplier of long-valued results.

29 **LongToDoubleFunction**

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

30 **LongToIntFunction**

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

31 **LongUnaryOperator**

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

32 **ObjDoubleConsumer<T>**

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

33 **ObjIntConsumer<T>**

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

34 **ObjLongConsumer<T>**

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

35 **Predicate<T>**

Represents a predicate $Boolean-valuedfunction$ of one argument.

36 **Supplier<T>**

Represents a supplier of results.

37 **ToDoubleBiFunction<T,U>**

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

38 **ToDoubleFunction<T>**

Represents a function that produces a double-valued result.

39 **ToIntBiFunction<T,U>**

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

40 **ToIntFunction<T>**

Represents a function that produces an int-valued result.

41 **ToLongBiFunction<T,U>**

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

42 **ToLongFunction<T>**

Represents a function that produces a long-valued result.

43 **UnaryOperator<T>**

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

Functional Interface Example

Predicate <T> interface is a functional interface with a method test*Object* to return a Boolean value. This interface signifies that an object is tested to be true or false.

To get more clarity on this, write the following program in an code editor and verify the results.

Java8Tester.java

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

public class Java8Tester {
   public static void main(String args[]){
      List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);

      // Predicate<Integer> predicate = n -> true
      // n is passed as parameter to test method of Predicate interface
      // test method will always return true no matter what value n has.

      System.out.println("Print all numbers:");

      //pass n as parameter
      eval(list, n->true);
```

```
      // Predicate<Integer> predicate1 = n -> n%2 == 0
      // n is passed as parameter to test method of Predicate interface
      // test method will return true if n%2 comes to be zero

      System.out.println("Print even numbers:");
      eval(list, n-> n%2 == 0 );

      // Predicate<Integer> predicate2 = n -> n > 3
      // n is passed as parameter to test method of Predicate interface
      // test method will return true if n is greater than 3.

      System.out.println("Print numbers greater than 3:");
      eval(list, n-> n > 3 );
   }

   public static void eval(List<Integer> list, Predicate<Integer> predicate) {
      for(Integer n: list) {

         if(predicate.test(n)) {
            System.out.println(n + " ");
         }
      }
   }
}
```

Here we've passed Predicate interface, which takes a single input and returns Boolean.

Verify the Result

Compile the class using **javac** compiler as follows −

```
$javac Java8Tester.java
```

Now run the Java8Tester as follows −

```
$java Java8Tester
```

It should produce the following output −

```
Print all numbers:
1
2
3
4
5
6
7
8
9
Print even numbers:
2
4
6
8
Print numbers greater than 3:
4
5
6
7
8
9
```