Vikash Singh
@full_stack_geek

Functional Interface in Java

Vikash Singh
@full_stack_geek

An Interface that contains exactly **one abstract method** is known as functional interface. It can have any number of default, static methods but can contain only one abstract method.Runnable, ActionListener, and Comparable are some of the examples of **functional interfaces.**

@FunctionalInterface annotation is used to ensure that the functional interface can’t have more than one abstract method.

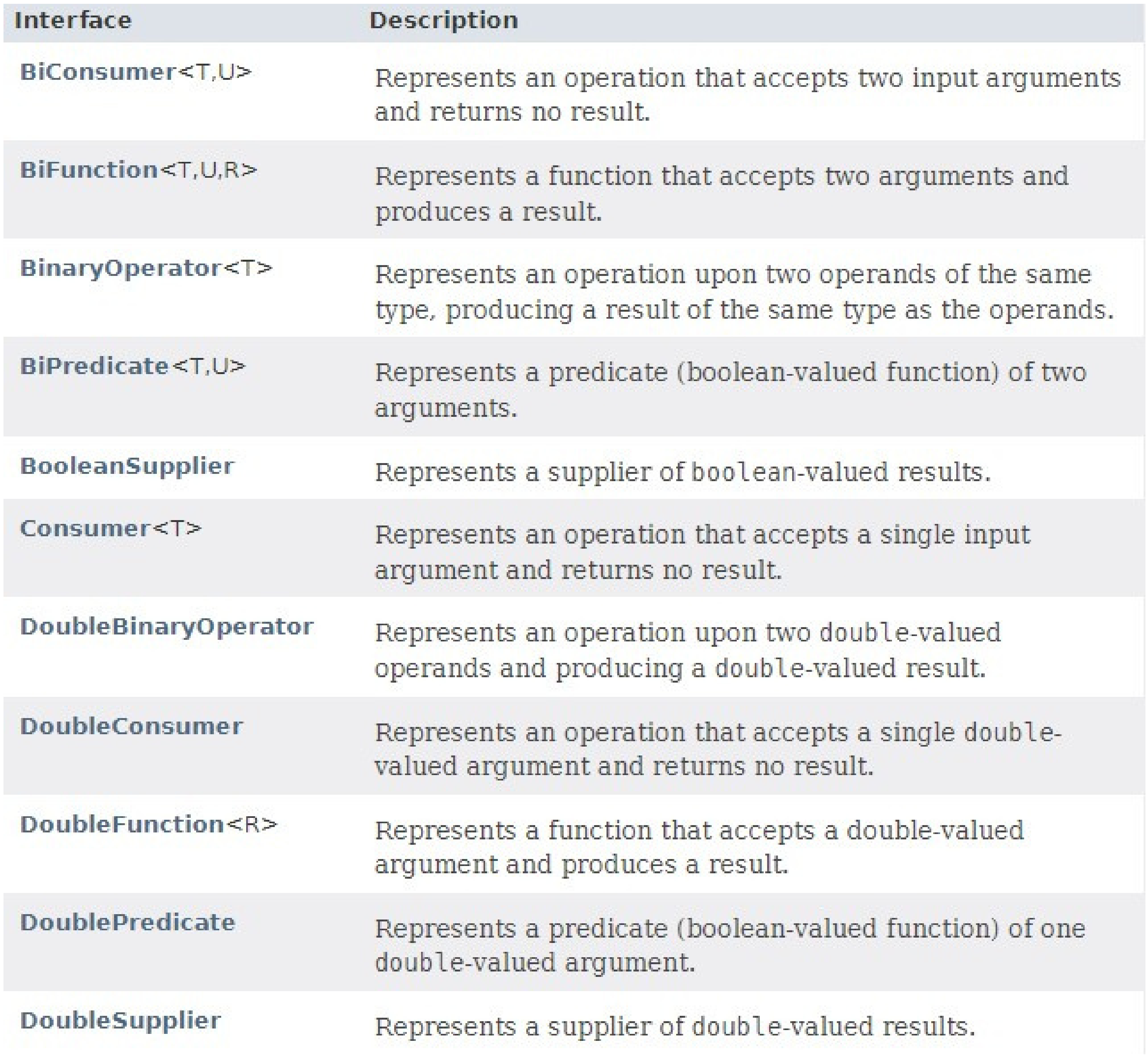
Vikash Singh
@full_stack_geek

Example :-

```
@FunctionalInterface
 interface Square {
   int calculate(int x);
}

class Test {
   public static void main(String args[])
   {
      int a = 10;

      // lambda expression to define the calculate method
      Square s = (int x) -> x * x;

      // parameter passed and return type must be
      // same as defined in the prototype
      int ans = s.calculate(a);
      System.out.println(ans);
   }
}
```

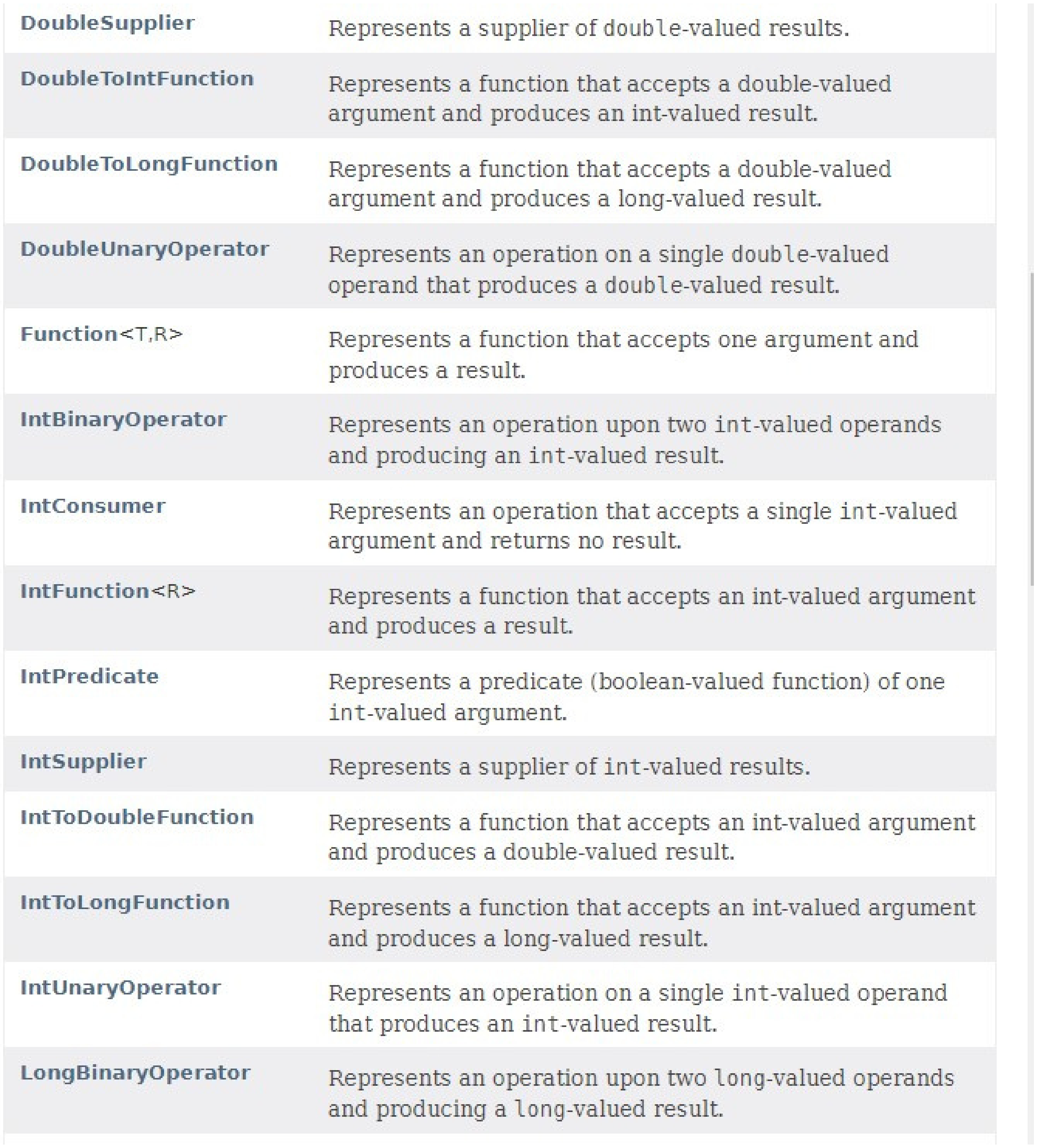
Vikash Singh
@full_stack_geek

After Java 8 , We have number of predefined functional interface.



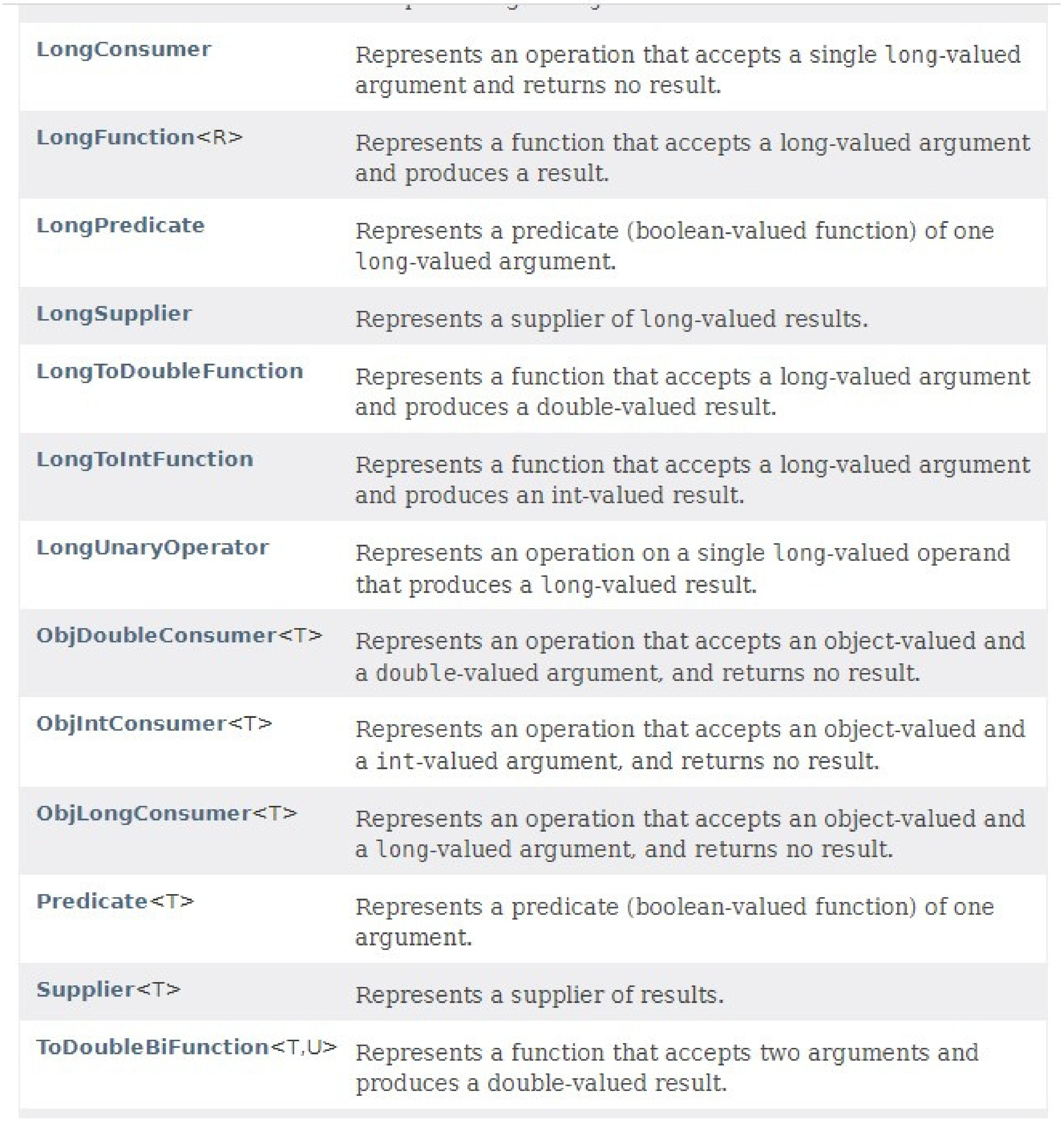
Interface	Description
BiConsumer<T,U>	Represents an operation that accepts two input arguments and returns no result.
BiFunction<T,U,R>	Represents a function that accepts two arguments and produces a result.
BinaryOperator<T>	Represents an operation upon two operands of the same type, producing a result of the same type as the operands.
BiPredicate<T,U>	Represents a predicate (boolean-valued function) of two arguments.
BooleanSupplier	Represents a supplier of boolean-valued results.
Consumer<T>	Represents an operation that accepts a single input argument and returns no result.
DoubleBinaryOperator	Represents an operation upon two double-valued operands and producing a double-valued result.
DoubleConsumer	Represents an operation that accepts a single double-valued argument and returns no result.
DoubleFunction<R>	Represents a function that accepts a double-valued argument and produces a result.
DoublePredicate	Represents a predicate (boolean-valued function) of one double-valued argument.
DoubleSupplier	Represents a supplier of double-valued results.

Vikash Singh
@full_stack_geek



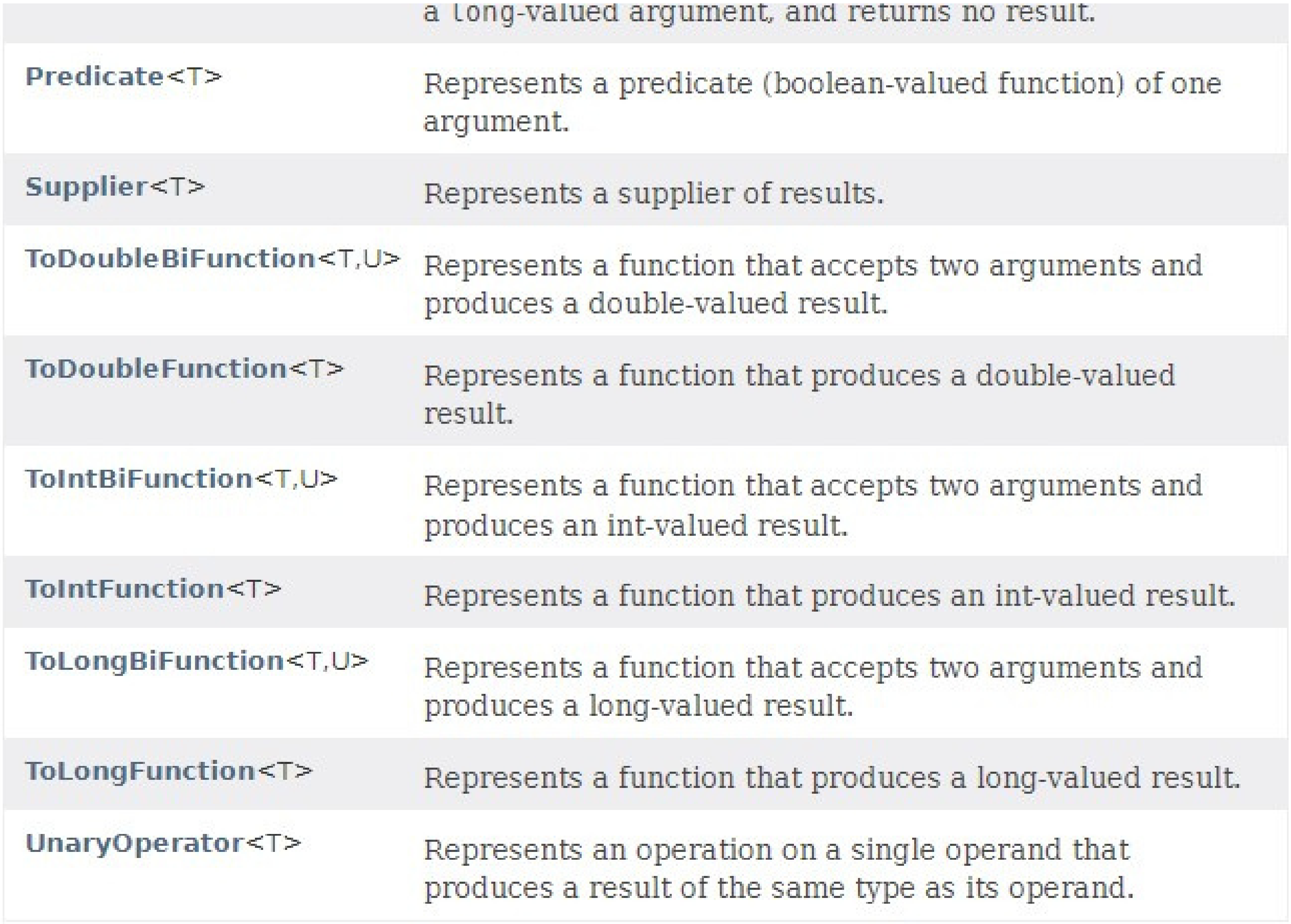
DoubleSupplier	Represents a supplier of double-valued results.
DoubleToIntFunction	Represents a function that accepts a double-valued argument and produces an int-valued result.
DoubleToLongFunction	Represents a function that accepts a double-valued argument and produces a long-valued result.
DoubleUnaryOperator	Represents an operation on a single double-valued operand that produces a double-valued result.
Function<T,R>	Represents a function that accepts one argument and produces a result.
IntBinaryOperator	Represents an operation upon two int-valued operands and producing an int-valued result.
IntConsumer	Represents an operation that accepts a single int-valued argument and returns no result.
IntFunction<R>	Represents a function that accepts an int-valued argument and produces a result.
IntPredicate	Represents a predicate (boolean-valued function) of one int-valued argument.
IntSupplier	Represents a supplier of int-valued results.
IntToDoubleFunction	Represents a function that accepts an int-valued argument and produces a double-valued result.
IntToLongFunction	Represents a function that accepts an int-valued argument and produces a long-valued result.
IntUnaryOperator	Represents an operation on a single int-valued operand that produces an int-valued result.
LongBinaryOperator	Represents an operation upon two long-valued operands and producing a long-valued result.

Vikash Singh
@full_stack_geek



LongConsumer	Represents an operation that accepts a single `long`-valued argument and returns no result.
LongFunction<R>	Represents a function that accepts a long-valued argument and produces a result.
LongPredicate	Represents a predicate (boolean-valued function) of one `long`-valued argument.
LongSupplier	Represents a supplier of `long`-valued results.
LongToDoubleFunction	Represents a function that accepts a long-valued argument and produces a double-valued result.
LongToIntFunction	Represents a function that accepts a long-valued argument and produces an int-valued result.
LongUnaryOperator	Represents an operation on a single `long`-valued operand that produces a `long`-valued result.
ObjDoubleConsumer<T>	Represents an operation that accepts an object-valued and a `double`-valued argument, and returns no result.
ObjIntConsumer<T>	Represents an operation that accepts an object-valued and a `int`-valued argument, and returns no result.
ObjLongConsumer<T>	Represents an operation that accepts an object-valued and a `long`-valued argument, and returns no result.
Predicate<T>	Represents a predicate (boolean-valued function) of one argument.
Supplier<T>	Represents a supplier of results.
ToDoubleBiFunction<T,U>	Represents a function that accepts two arguments and produces a double-valued result.

Vikash Singh
@full_stack_geek



	a long-valued argument, and returns no result.
Predicate<T>	Represents a predicate (boolean-valued function) of one argument.
Supplier<T>	Represents a supplier of results.
ToDoubleBiFunction<T,U>	Represents a function that accepts two arguments and produces a double-valued result.
ToDoubleFunction<T>	Represents a function that produces a double-valued result.
ToIntBiFunction<T,U>	Represents a function that accepts two arguments and produces an int-valued result.
ToIntFunction<T>	Represents a function that produces an int-valued result.
ToLongBiFunction<T,U>	Represents a function that accepts two arguments and produces a long-valued result.
ToLongFunction<T>	Represents a function that produces a long-valued result.
UnaryOperator<T>	Represents an operation on a single operand that produces a result of the same type as its operand.

Vikash Singh
@full_stack_geek

Example:-

```
import java.util.function.BiConsumer;
public class FunctionalInterfaceExample {
    static void ShowDetails(String name, Integer age){
        System.out.println(name+" "+age);
    }
    public static void main(String[] args) {
        // Referring method
        BiConsumer<String, Integer> biCon =
BiConsumerInterfaceExample::ShowDetails;
        biCon.accept("Rama", 20);

   Predicate<Integer> pr = a -> (a > 18); // Creating predicate
        System.out.println(pr.test(10));    // Calling Predicate method

    // Function interface referring to a method
      Function<String, String> fun = FunctionInterfaceExample::ShowDetails;
        // Calling Function interface method
        System.out.println(fun.apply("Peter"));
    }
}
```