

UnaryOperator<T>#

The `UnaryOperator<T>` interface represents a function that takes one argument of type `T` and returns a value of the same type. This is similar to the `Function` interface, which is a parent to the `UnaryOperator` interface.

The `UnaryOperator` does not define any new abstract methods. Since it extends the `Function` interface from the same package, it inherits the following method from the `Function` interface :

```
1 T apply(T t)
```

Below are the functional interfaces, which can be categorized as unary operators

<code>UnaryOperator<T></code>	Represents an operation on a single operand that produces a result of the same type as its operand (reference type)	<code>T apply (T t)</code>
<code>DoubleUnaryOperator</code>	Accepts single double-value operand and produces a double-value result	<code>double applyAsDouble(double operand)</code>
<code>IntUnaryOperator</code>	Accepts a single int-value operand and produces an int-value result	<code>int applyAsInt(int operand)</code>
<code>LongUnaryOperator</code>	Accepts a single long-value operand and produces a long-value result	<code>long applyAsLong(long operand)</code>

In the below example, we will create a lambda of unary operator type. It will take a `Person` object as input, fill data in the object, and return the same object as the output.

```
1 import java.util.function.UnaryOperator;
2
3 public class UnaryOperatorTest {
4
5
6     public static void main(String args[]) {
7         Person person = new Person();
8         UnaryOperator<Person> operator = (p) -> {
9             p.name = "John";
10            p.age = 34;
11            return p;
12        };
13
14        operator.apply(person);
15        System.out.println("Person Name: " + person.getName() + " Person Age: " + person.getAge());
16    }
17 }
18
19 class Person {
20     String name;
21     int age;
22
23     Person() {
24     }
25
26     Person(String name, int age) {
27         this.name = name;
28         this.age = age;
29     }
30 }
```

Run Save Reset

IntUnaryOperator#

This is the primitive flavor of the `UnaryOperator` . It takes an `int` as an argument and returns `int` as a result. We should always prefer using the primitive flavors of functional interfaces as boxing and unboxing are not good for performance.

In the below example, we will create a lambda of `IntUnaryOperator` type. It takes a number as input and returns its square.

```
1 import java.util.function.IntUnaryOperator;
2
3 public class UnaryOperatorTest {
4
5
6     public static void main(String args[]) {
7         IntUnaryOperator operator = num -> num * num;
8
9         System.out.println(operator.applyAsInt(25));
10    }
11 }
12
```

Run Save Reset