`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 :

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

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

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

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.

```
import java.util.function.UnaryOperator;

public class UnaryOperatorTest {


    public static void main(String args[]) {
        Person person = new Person();
        UnaryOperator<Person> operator = (p) -> {
            p.name = "John";
            p.age = 34;
            return p;
        };

        operator.apply(person);
        System.out.println("Person Name: " + person.getName() + " Person Age: " + person.getAge());
    }
}

class Person {
    String name;
    int age;

    Person() {
    }

    Person(String name, int age) {
        this.name = name;
        this.age = age;
```

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.

```
import java.util.function.IntUnaryOperator;

public class UnaryOperatorTest {


    public static void main(String args[]) {
        IntUnaryOperator operator = num -> num * num;

        System.out.println(operator.applyAsInt(25));
    }
}

```

Run Save Reset

In the next lesson, we will discuss the `Binary` operator.