IntUnaryOperator Functional Interface

Before we discuss IntUnaryOperator I would like to point out that this interface is primitive type specialization of [UnaryOperator](http://data-structure-learning.blogspot.com/2015/07/java-lambda-unaryoperator-functional.html). Also, UnaryOperator Interface extends [Function](http://data-structure-learning.blogspot.com/2015/07/java-lambda-function-functional.html) interface. I highly recommend that you read those interfaces.

IntUnaryOperator interface represents operation on int valued operand that produces int valued result.

Below are the explanations of methods in IntUnaryOperator interface.

**applyAsInt() method**

**int** applyAsInt(**int** operand);

applyAsInt() method applies this operator to operand passed as parameter in method.

IntUnaryOperator power = (val) -> val + val;

System.***out***.println(power.applyAsInt(10)); //Output 20

**compose() method**

**default** IntUnaryOperator compose(IntUnaryOperator before) {

Objects.*requireNonNull*(before);

**return** (**int** v) -> applyAsInt(before.applyAsInt(v));

}

Compose method is used to return a composed operator that applies before operator (as in parameter) first and then applies this operator to the result. If the operator passed in parameter is null then this method will throw NullPointerException. Let us take simple example of addition and subtractions using IntUnaryOperator.

IntUnaryOperator substracter = (val) -> {

val = val - 10;

System.***out***.println("substracter " + val);

**return** val;

};

IntUnaryOperator adder = (val) -> {

val = val + 13;

System.***out***.println("adder " + val);

**return** val;

};

adder.compose(substracter).applyAsInt(33);

Output

substracter 23

adder 36

**andThen() method**

**default** IntUnaryOperator andThen(IntUnaryOperator after) {

Objects.*requireNonNull*(after);

**return** (**int** t) -> after.applyAsInt(applyAsInt(t));

}

andThen() method is used to return the composed operator that applies this operator first and then it applies after operator to the result of this operator. If the operator passed in parameter is null then this method will throw NullPointerException. Let us take simple example of addition and subtractions using IntUnaryOperator.

IntUnaryOperator substracter = (val) -> {

val = val - 10;

System.***out***.println("substracter " + val);

**return** val;

};

IntUnaryOperator adder = (val) -> {

val = val + 13;

System.***out***.println("adder " + val);

**return** val;

};

adder.andThen(substracter).applyAsInt(33);

Output

adder 46

substracter 36

**identity() method**

**static** IntUnaryOperator identity() {

**return** t -> t;

}

identity() method returns a unary operator that will always return the input argument.

IntUnaryOperator identity = IntUnaryOperator.*identity*();

System.***out***.println(identity.applyAsInt(10));

Output

10

That’s all on IntUnaryOperator interface.

Read about important java.util.function package’s interface [here](http://data-structure-learning.blogspot.com/p/functional-programming-in-java.html). [Consumer](http://data-structure-learning.blogspot.com/2015/07/java-lambda-consumer-functional.html), [Function](http://data-structure-learning.blogspot.com/2015/07/java-lambda-function-functional.html), [Supplier](http://data-structure-learning.blogspot.com/2015/07/java-lambda-supplier-functional.html), [BinaryOperator](http://data-structure-learning.blogspot.com/2015/07/java-lambda-binaryoperator-functional.html) & [Predicate](http://data-structure-learning.blogspot.com/2015/07/java-lambda-predicate-functional.html) Functional Interfaces. I have also written on [High Order functions](http://data-structure-learning.blogspot.com/2015/07/higher-order-functions-using-function.html) using Function functional interface.