DoubleUnaryOperator Functional Interface

Before we discuss DoubleUnaryOperator 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.

DoubleUnaryOperator interface represents operation on double valued operand that produces double valued result.

Below are the explanations of methods in DoubleUnaryOperator interface.

|  |
| --- |
| **applyAsDouble() method**  **double** applyAsDouble(**double** operand);  applyAsDouble() method applies this operator to operand passed as parameter in method.  DoubleUnaryOperator power = (val) -> Math.*pow*(val, 2);  System.***out***.println(power.applyAsDouble(10.0)); //Outputs 100.0 |
| **compose() method**  **default** DoubleUnaryOperator compose(DoubleUnaryOperator before) {  Objects.*requireNonNull*(before);  **return** (**double** v) -> applyAsDouble(before.applyAsDouble(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 DoubleUnaryOperator.  DoubleUnaryOperator subtracter = (val) -> {  val = val - 10.9;  System.***out***.println("subtracter " + val);  **return** val;  };  DoubleUnaryOperator adder = (val) -> {  val = val + 10.5;  System.***out***.println("adder " + val);  **return** val;  };  adder.compose(subtracter).applyAsDouble(33);  Output  subtracter 22.1  adder 32.6 |
| **andThen() method**  **default** DoubleUnaryOperator andThen(DoubleUnaryOperator after) {  Objects.*requireNonNull*(after);  **return** (**double** t) -> after.applyAsDouble(applyAsDouble(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 DoubleUnaryOperator.  DoubleUnaryOperator substracter = (val) -> {  val = val - 10.9;  System.***out***.println("substracter " + val);  **return** val;  };  DoubleUnaryOperator adder = (val) -> {  val = val + 10.5;  System.***out***.println("adder " + val);  **return** val;  };  adder.andThen(substracter).applyAsDouble(33);  Output  adder 43.5  substracter 32.6 |
| **identity() method**  **static** DoubleUnaryOperator identity() {  **return** t -> t;  }  identity() method returns a unary operator that will always return the input argument.  DoubleUnaryOperator identity=DoubleUnaryOperator.*identity*();  System.***out***.println(identity.applyAsDouble(10));  Output  10 |

That’s all on DoubleUnaryOperator 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.