UnaryOperator Functional Interface

UnaryOperator interface represents an operation on single operand that produces a result of same type of operand. We have discussed Function, BiFunction, DoubleFunction, DoubleToIntFunction, DoubleToLongFunction interfaces. I would highly recommend reading them.

UnaryOperator interface extends Function interface. Function interfaces accepts arguments of any type T and returns result of any type R. Contrary to that UnaryOperator accepts input of type T and return result of type T too. So in case of UnaryOperator the input and result type are same whereas in Function they are different (they can be same too).

UnaryOperator is specialization of Function interface. Also UnaryOperator extends Function interface.

|  |
| --- |
| **UnaryOperator Interface Declaration**  **public** **interface** UnaryOperator<T> **extends** Function<T, T>  So UnaryOperator of type T extends Function interface of type T. Notice input and result type in Function interface are same. Hence, UnaryOperator is specialization of Function interface. |
| **apply() method (Function interface)**  UnaryOperator<String> unaryOp = (str) -> str.toLowerCase();  String str = unaryOp.apply("JAVA");  System.***out***.println(str); //Outputs java |
| **andThen() method (Function interface)**  UnaryOperator<String> toLowerCase = (str) -> {  String result = str.toLowerCase();  System.***out***.println(result); //Outputs java version  **return** result;  };  UnaryOperator<String> toUpperCase = (str) -> {  String result = str.toUpperCase(); //Outputs JAVA VERSION  System.***out***.println(result);  **return** result;  };  toLowerCase.andThen(toUpperCase).apply("JaVa VeRsIoN");  Outputs  java version  JAVA VERSION |
| **compose() method (Function interface)**  UnaryOperator<String> toLowerCase = (str) -> {  String result = str.toLowerCase();  System.***out***.println(result); //Outputs java version  **return** result;  };  UnaryOperator<String> toUpperCase = (str) -> {  String result = str.toUpperCase(); //Outputs JAVA VERSION  System.***out***.println(result);  **return** result;  };  toLowerCase.compose(toUpperCase).apply("JaVa VeRsIoN");  Outputs  JAVA VERSION  java version |
| **identity() method (UnaryOperator interface’s default method)**  **static** <T> UnaryOperator<T> identity() {  **return** t -> t;  }  This method returns unary operator that always returns its input argument.  UnaryOperator<String> unaryOp = UnaryOperator.*identity*();  System.***out***.println(unaryOp.apply("Operator"));//Outputs Operator |

That’s all on UnaryOperator.

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.