BinaryOperator Functional Interface

BinaryOperator interface extends [BiFunction interface](http://data-structure-learning.blogspot.com/2015/07/java-lambda-bifunction-functional.html) and BiFunction interface uses [Function interface](http://data-structure-learning.blogspot.com/2015/07/higher-order-functions-using-function.html). So this post is heavily dependent on those posts. Kindly refer them all.

|  |
| --- |
| BinaryOperator Interface Declaration  **public** **interface** BinaryOperator<T> **extends** BiFunction<T,T,T>  So BinaryOperator interface extends BiFunction interface. Now when we saw the BiFunction interface it had 3 different arguments *T*, *U* and *R*. But over here it has 3 arguments of same type *T*. The reason behind it is that BinaryOperator accepts only parameter of type *T*. |
| apply() and andThen() method  This both methods are inherited from BiFunction interface. I have written about those method in [BiFunction Interface](http://data-structure-learning.blogspot.com/2015/07/java-lambda-bifunction-functional.html) post. |
| minBy() method  **public** **static** <T> BinaryOperator<T> minBy(Comparator<? **super** T> comparator) {  Objects.*requireNonNull*(comparator);  **return** (a, b) -> comparator.compare(a, b) <= 0 ? a : b;  }  This method is used to find min of 2 objects that is supplied in apply() method.  Comparator must not be null else it will throw NullPointerException.  This method returns BinaryOperator which returns min of 2 elements as specified by Comparator. |
| maxBy() method  **public** **static** <T> BinaryOperator<T> maxBy(Comparator<? **super** T> comparator) {  Objects.*requireNonNull*(comparator);  **return** (a, b) -> comparator.compare(a, b) >= 0 ? a : b;  }  This method is used to find max of 2 objects that is supplied in apply() method.  Comparator must not be null else it will throw NullPointerException.  This method returns BinaryOperator which returns max of 2 elements as specified by Comparator. |

We can also use method reference in Comparator as Comparator’s compare method expects integer result.

Using apply() method for addition.

BinaryOperator<Integer> biOp = (t, u) -> t + u;

**int** result = biOp.apply(2, 3);

System.***out***.println("apply() method "+result);//Outputs 5

|  |
| --- |
| Using minBy() method  For these method we will use Comparator.reverseComparator() as well as static method reference Integer::*compare.*  First we will use Integer::*compare*  BinaryOperator<Integer> biOpMin = BinaryOperator.*minBy*(Integer::*compare*);  **int** min = biOpMin.apply(12, 33);  System.***out***.println("minBy() "+min);//Outputs 12  Let us now use Comparator.reverseComparator(). Reverse comparator is used to sort in reverse order.  BinaryOperator<Integer> biOpMin = BinaryOperator.*minBy*(Comparator.*reverseOrder*());  **int** min = biOpMin.apply(12, 33);  System.***out***.println("minBy() "+min);//Outputs 33 |
| Using maxBy() method  For this method too we will use static method reference Integer::*compare* and Comparator.reverseComparator()  First we will use Integer::*compare*  BinaryOperator<Integer> biOpMax = BinaryOperator.*maxBy*(Integer::*compare*);  **int** max = biOpMax.apply(12, 33);  System.***out***.println("maxBy() "+max); //Outputs 33    Now we will use Comparator.reverseorder()  BinaryOperator<Integer> biOpMax = BinaryOperator.*maxBy*(Comparator.*reverseOrder*());  **int** max = biOpMax.apply(12, 33);  System.***out***.println("minBy() "+max); //Outputs 12 |