BiPredicate Functional Interface

BiPredicate interface is similar to Predicate interface. We have seen Predicate interface basics, test() method, and() method, negate()method, or() method and reusing Predicates.

BiPredicate represents a predicate or boolean-valued function of two arguments. Predicate represents a boolean-valued function of single or one argument.

|  |
| --- |
| **BiPredicate Interface Declaration**  **public** **interface** BiPredicate<T, U>  *T* and *U* are two input arguments. |
| **test() method**  **boolean** test(T t, U u);  test() method is used to evaluate this predicate on given arguments.  BiPredicate<Integer, Integer> biPredGreater = (t, u) -> t > u;  System.***out***.println(biPredGreater.test(10, 9)); //Outputs *true*.  BiPredicate<String, String> biPredEquals = (t, u) -> t.equals(u);  System.***out***.println(biPredEquals.test("Orion Pax", "Megatronus"));//Outputs *false* |
| **and() method**  **default** BiPredicate<T, U> and(BiPredicate<? **super** T, ? **super** U> other) {  Objects.*requireNonNull*(other);  **return** (T t, U u) -> test(t, u) && other.test(t, u);  }  and() method returns a composed predicate that represents the short-circuiting logical AND of this and *other* predicate. While evaluating if this predicate’s test() method returns false then *other* predicate is not evaluated.  and() method will throw NullPointerException if *other* BiPredicate is null.  We will write two different BiPredicate (to check inequality for String contents and String length)  BiPredicate<String, String> biPredNotEquals = (t, u) -> !t.equals(u);    BiPredicate<String, String> biPredLengthChecker = (t, u) -> t.length() > u.length();  Now let us built a composed version of this both BiPredicate. This BiPredicate will return true of both the arguments contents are not same and length of first argument is greater than second.  BiPredicate<String, String> biPred = **biPredNotEquals.and(biPredLengthChecker);**  **boolean** test = biPred.test("AB", "AB");  System.***out***.println(test); //Outputs false  test = biPred.test("AB", "ABC");  System.***out***.println(test); //Outputs false  test = biPred.test("ABC", "AB");  System.***out***.println(test); //Outputs true |
| **negate() method**  **default** BiPredicate<T, U> negate() {  **return** (T t, U u) -> !test(t, u);  }  This method returns the predicate that represents the logical negation of this predicate.  BiPredicate<String, String> biPredEquals = (t, u) -> t.equals(u);  System.***out***.println(biPredEquals.test("AA", "AA")); //Outputs *true*  BiPredicate<String, String> biPredNegate = **biPredEquals.negate()**;  System.***out***.println(biPredNegate.test("AA", "AA")); //Outputs false |
| **or() method**  **default** BiPredicate<T, U> or(BiPredicate<? **super** T, ? **super** U> other) {  Objects.*requireNonNull*(other);  **return** (T t, U u) -> test(t, u) || other.test(t, u);  }  or() method demo  BiPredicate<Integer, Integer> biPred1 = (t, u) -> t > 50 && u < 60;  BiPredicate<Integer, Integer> biPred2 = (t, u) -> t > 55 && u < 65;  BiPredicate<Integer, Integer> or = **biPred1.or(biPred2);**  **boolean** test = or.test(40, 40); //Outputs false  System.***out***.println(test);  test = or.test(60, 55); //Outputs true  System.***out***.println(test);  That’s all on BiPredicate interface. |