

JAVA 8

(Bi) Functional Interfaces

Two-Argument (Bi) Functional Interfaces

Need of Two-Argument (Bi) Functional Interfaces:

Normal Functional Interfaces (Predicate, Function and Consumer) can accept only one input argument. But sometimes our programming requirement is to accept two input arguments, then we should go for two-argument functional interfaces. The following functional interfaces can take 2 input arguments.

1. BiPredicate
2. BiFunction
3. BiConsumer

1. BiPredicate(I):

Normal Predicate can take only one input argument and perform some conditional check. Sometimes our programming requirement is we have to take 2 input arguments and perform some conditional check, for this requirement we should go for BiPredicate.

BiPredicate is exactly same as Predicate except that it will take 2 input arguments.

```
1) interface BiPredicate<T1,T2>
2) {
3)     public boolean test(T1 t1,T2 t2);
4)     //remaining default methods: and(), or() , negate()
5) }
```

To check the sum of 2 given integers is even or not by using BiPredicate:

```
1) import java.util.function.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         BiPredicate<Integer,Integer> p=(a,b)->(a+b) %2==0;
7)         System.out.println(p.test(10,20));
8)         System.out.println(p.test(15,20));
9)     }
10) }
```

Output:

true
false

2.BiFunction:

Normal Function can take only one input argument and perform required operation and returns the result. The result need not be boolean type.

But sometimes our programming requirement to accept 2 input values and perform required operation and should return the result. Then we should go for BiFunction.

BiFunction is exactly same as funtion except that it will take 2 input arguments.

```
1) interface BiFunction<T,U,R>
2) {
3)     public R apply(T t,U u);
4)     //default method andThen()
5) }
```

To find product of 2 given integers by using BiFunction:

```
1) import java.util.function.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         BiFunction<Integer,Integer,Integer> f=(a,b)->a*b;
7)         System.out.println(f.apply(10,20));
8)         System.out.println(f.apply(100,200));
9)     }
10) }
```

To create Student Object by taking name and rollno as input by using BiFunction:

```
1) import java.util.function.*;
2) import java.util.*;
3) class Student
4) {
5)     String name;
6)     int rollno;
7)     Student(String name,int rollno)
8)     {
9)         this.name=name;
10)        this.rollno=rollno;
11)    }
12) }
13) class Test
14) {
```

```

15) public static void main(String[] args)
16) {
17)     ArrayList<Student> l = new ArrayList<Student>();
18)     BiFunction<String,Integer,Student> f=(name,rollno)->new Student(name,rollno);
19)
20)     l.add(f.apply("Durga",100));
21)     l.add(f.apply("Ravi",200));
22)     l.add(f.apply("Shiva",300));
23)     l.add(f.apply("Pavan",400));
24)     for(Student s : l)
25)     {
26)         System.out.println("Student Name:"+s.name);
27)         System.out.println("Student Rollno:"+s.rollno);
28)         System.out.println();
29)     }
30) }
31) }

```

Output:

Student Name:Durga
Student Rollno:100

Student Name:Ravi
Student Rollno:200

Student Name:Shiva
Student Rollno:300

Student Name:Pavan
Student Rollno:400

To calculate Monthly Salary with Employee and TimeSheet objects as input By using BiFunction:

```

1) import java.util.function.*;
2) import java.util.*;
3) class Employee
4) {
5)     int eno;
6)     String name;
7)     double dailyWage;
8)     Employee(int eno,String name,double dailyWage)
9)     {
10)         this.eno=eno;
11)         this.name=name;
12)         this.dailyWage=dailyWage;
13)     }

```

```

14) }
15) class TimeSheet
16) {
17)     int eno;
18)     int days;
19)     TimeSheet(int eno,int days)
20)     {
21)         this.eno=eno;
22)         this.days=days;
23)     }
24) }
25) class Test
26) {
27)     public static void main(String[] args)
28)     {
29)         BiFunction<Employee,TimeSheet,Double> f=(e,t)->e.dailyWage*t.days;
30)         Employee e= new Employee(101,"Durga",1500);
31)         TimeSheet t= new TimeSheet(101,25);
32)         System.out.println("Employee Monthly Salary:"+f.apply(e,t));
33)     }
34) }

```

Output: Employee Monthly Salary:37500.0

BiConsumer:

Normal Consumer can take only one input argument and perform required operation and won't return any result.

But sometimes our programming requirement to accept 2 input values and perform required operation and not required to return any result. Then we should go for BiConsumer.

BiConsumer is exactly same as Consumer except that it will take 2 input arguments.

```

1) interface BiConsumer<T,U>
2) {
3)     public void accept(T t,U u);
4)     //default method andThen()
5) }

```

Program to accept 2 String values and print result of concatenation by using BiConsumer:

```
1) import java.util.function.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         BiConsumer<String,String> c=(s1,s2)->System.out.println(s1+s2);
7)         c.accept("durga","soft");
8)     }
9) }
```

Output: durgasoft

Demo Program to increment employee Salary by using BiConsumer:

```
1) import java.util.function.*;
2) import java.util.*;
3) class Employee
4) {
5)     String name;
6)     double salary;
7)     Employee(String name,double salary)
8)     {
9)         this.name=name;
10)        this.salary=salary;
11)    }
12) }
13) class Test
14) {
15)     public static void main(String[] args)
16)     {
17)         ArrayList<Employee> l= new ArrayList<Employee>();
18)         populate(l);
19)         BiConsumer<Employee,Double> c=(e,d)->e.salary=e.salary+d;
20)         for(Employee e:l)
21)         {
22)             c.accept(e,500.0);
23)         }
24)         for(Employee e:l)
25)         {
26)             System.out.println("Employee Name:"+e.name);
27)             System.out.println("Employee Salary:"+e.salary);
28)             System.out.println();
29)         }
30) }
```

```

31) }
32) public static void populate(ArrayList<Employee> l)
33) {
34)     l.add(new Employee("Durga",1000));
35)     l.add(new Employee("Sunny",2000));
36)     l.add(new Employee("Bunny",3000));
37)     l.add(new Employee("Chinny",4000));
38) }
39) }

```

Comparison Table between One argument and Two argument Functional Interfaces:

One Argument Functional Interface	Two Argument Functional Interface
<pre> interface Predicate<T> { public boolean test(T t); default Predicate and(Predicate P) default Predicate or(Predicate P) default Predicate negate() static Predicate isEqual(Object o) } </pre>	<pre> interface BiPredicate<T, U> { public boolean test(T t, U u); default BiPredicate and(BiPredicate P) default BiPredicate or(BiPredicate P) default BiPredicate negate() } </pre>
<pre> interface Function<T, R> { public R apply(T t); default Function andThen(Function F) default Function compose(Function F) static Function identify() } </pre>	<pre> interface BiFunction<T, U, R> { public R apply(T t, U u); default BiFunction andThen(Function F) } </pre>
<pre> interface Consumer<T> { public void accept(T t); default Consumer andThen(Consumer C) } </pre>	<pre> interface BiConsumer<T, U> { public void accept(T t, U u); default BiConsumer andThen(BiConsumer C) } </pre>