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.

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

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

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

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.

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

To find product of 2 given integers by using BiFunction:

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

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

```
import java.util.function.*;
import java.util.*;
class Student
{
   String name;
   int rollno;
   Student(String name,int rollno)
   {
      this.name=name;
      this.rollno=rollno;
   }
}
class Test
{
```

```
    public static void main(String[] args)
    {
        ArrayList<Student> l = new ArrayList<Student>();
        BiFunction<String,Integer,Student> f=(name,rollno)->new Student(name,rollno);

        l.add(f.apply("Durga",100));
        l.add(f.apply("Ravi",200));
        l.add(f.apply("Shiva",300));
        l.add(f.apply("Pavan",400));
        for(Student s : l)
        {
            System.out.println("Student Name:"+s.name);
            System.out.println("Student Rollno:"+s.rollno);
            System.out.println();
        }
    }
}
```

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:

```
import java.util.function.*;
import java.util.*;
class Employee
{
    int eno;
    String name;
    double dailyWage;
    Employee(int eno,String name,double dailyWage)
    {
        this.eno=eno;
        this.name=name;
        this.dailyWage=dailyWage;
    }
}
class TimeSheet
{
   int eno;
   int days;
   TimeSheet(int eno,int days)
   {
      this.eno=eno;
      this.days=days;
   }
}
class Test
{
   public static void main(String[] args)
   {
      BiFunction<Employee,TimeSheet,Double> f=(e,t)->e.dailyWage*t.days;
      Employee e= new Employee(101,"Durga",1500);
      TimeSheet t= new TimeSheet(101,25);
      System.out.println("Employee Monthly Salary:"+f.apply(e,t));
   }
}
```

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.

```
interface BiConsumer<T,U>
{
   public void accept(T t,U u);
   //default method andThen()
}
```

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

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

Output: durgasoft

Demo Program to increment employee Salary by using BiConsumer:

```java
import java.util.function.*;
import java.util.*;
class Employee
{
   String name;
   double salary;
   Employee(String name,double salary)
   {
      this.name=name;
      this.salary=salary;
   }
}
class Test
{
   public static void main(String[] args)
   {
      ArrayList<Employee> l= new ArrayList<Employee>();
      populate(l);
      BiConsumer<Employee,Double> c=(e,d)->e.salary=e.salary+d;
      for(Employee e:l)
      {
         c.accept(e,500.0);
      }
      for(Employee e:l)
      {
         System.out.println("Employee Name:"+e.name);
         System.out.println("Employee Salary:"+e.salary);
         System.out.println();
      }

```

```
    }
    public static void populate(ArrayList<Employee> l)
    {
       l.add(new Employee("Durga",1000));
       l.add(new Employee("Sunny",2000));
       l.add(new Employee("Bunny",3000));
       l.add(new Employee("Chinny",4000));
    }
}
```

Comparison Table between One argument and Two argument Functional Interfaces:

One Argument Functional Interface	Two Argument Functional Interface

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)
}
	interface BiPredicate<T, U>
	{
	public boolean test(T t, U u);
	default BiPredicate and(BiPredicate P)
	default BiPredicate or(BiPredicate P)
	default BiPredicate negate()
	}

interface Function<T, R>
{
public R apply(T t);
default Function andThen(Function F)
default Function compose(Function F)
static Function identify()
}
	interface BiFunction<T, U, R>
	{
	public R apply(T t, U u);
	default BiFunction andThen(Function F)
	}

interface Consumer<T>
{
public void accept(T t);
default Consumer andThen(Consumer C)
}
	interface BiConsumer<T, U>
	{
	public void accept(T t, U u);
	default BiConsumer andThen(BiConsumer C)
	}