**INTERNITY FOUNDATION**

**TASK-6**

**Submitted By:**

**Amisha Singhal**

**Java Batch**

**Lambda Built-in Functional Interfaces**

**Ans-** A *functional interface* in Java is an interface that contains only a single abstract (unimplemented) method. A functional interface can contain default and static methods which do have an implementation, in addition to the single unimplemented method.

**Eg:**

**class Test**

**{**

**public static void main(String args[])**

**{**

**// create anonymous inner class object**

**new Thread(new Runnable()**

**{**

**@Override**

**public void run()**

**{**

**System.out.println("New thread created");**

**}**

**}).start();**

**}**

**}  
Use the built-in interfaces included in the java.util.function package such as Predicate, Consumer, Function, and Supplier**

**Ans-**

**Predicate**

A predicate is a statement that may be true or false depending on the values of its variables. It can be thought of as a function that returns a value that is either true or false. A Predicate is a functional interface that can be used anywhere you need to evaluate a boolean condition. Since it's a functional interface, you can pass a lambda expression wherever a Predicate is expected.

**Eg-**

**Predicate<String> isALongWord = new Predicate<String>() {**

**@Override**

**public boolean test(String t) {**

**return t.length() > 10;**

**}**

**};**

**String s = "successfully"**

**boolean result = isALongWord.test(s);**

## Function

The Java Function interface (java.util.function.Function) interface is one of the most central functional interfaces in Java. The Function interface represents a function (method) that takes a single parameter and returns a single value.

**Eg-** **public class Test {**

**public static void main(String[] args) {**

**int n = 5;**

**modifyTheValue(n, val-> val + 10);**

**modifyTheValue(n, val-> val \* 100);**

**}**

**static void modifyValue(int v, Function<Integer, Integer> function){**

**int result = function.apply(v);**

**System.out.println(newValue);**

**}}**

## Consumer

The Java Consumer interface is a functional interface that represents a function that consumes a value without returning any value. A Java Consumer implementation could be printing out a value, or writing it to a file, or over the network etc.

**Eg-**

**Consumer<Integer> consumer = (value) -> System.out.println(value);**

This Java Consumer implementation prints the value passed as parameter to it out to System.out.

### Supplier

This functional interface does the opposite of the Consumer, it takes no arguments but it returns some value. It may return different values when it is being called more than once. Since it's a functional interface, we can pass a lambda expression wherever a Supplier is expected.

**Eg-**

**public class Program {**

**public static void main(String[] args) {**

**int n = 3;**

**display(() -> n + 10);**

**display(() -> n + 100);**

**}**

**static void display(Supplier<Integer> arg) {**

**System.out.println(arg.get());**

**}**

**}**

**Develop code that uses primitive versions of functional interfaces.**

**Ans**- If we want to use primitive types with these functional interfaces, Java uses a mechanism called auto-boxing to automatically convert a primitive to its corresponding wrapper type (for example, int to Integer) and vice versa.

But since boxed values use more memory, this comes with a performance cost. For this reason, Java provides specialized versions of the functional interfaces to avoid auto-boxing operations when the inputs or outputs are primitives.

For example, instead of using

**Predicate<Integer> p = i -> i > 10;**

You can use

**IntPredicate p = i -> i > 10;**

Primitive version of functional interfaces is as follows:

**Predicate<T>**  
[IntPredicate](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntPredicate.html):Predicate of one int-valued argument.  
[LongPredicate](https://docs.oracle.com/javase/8/docs/api/java/util/function/LongPredicate.html):Predicate of one long-valued argument.  
[DoublePredicate](https://docs.oracle.com/javase/8/docs/api/java/util/function/DoublePredicate.html): Predicate of one double-valued argument.

**Consumer<T>**  
[IntConsumer](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntConsumer.html):Operation that accepts a single int-valued argument and returns no result.  
[LongConsumer](https://docs.oracle.com/javase/8/docs/api/java/util/function/LongConsumer.html):Operation that accepts a single long-valued argument and returns no result.  
[DoubleConsumer](https://docs.oracle.com/javase/8/docs/api/java/util/function/DoubleConsumer.html):Operation that accepts a single double-valued argument and returns no result.

**Function<T, R>**  
[IntFunction<R>](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntFunction.html):Function that accepts an int-valued argument and produces a result.  
[IntToDoubleFunction](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntToDoubleFunction.html): Function that accepts an int-valued argument and produces a double-valued result.  
[IntToLongFunction](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntToLongFunction.html): Function that accepts an int-valued argument and produces a long-valued result.  
LongFunction<R>: Function that accepts a long-valued argument and produces a result.

**Supplier<T>**  
[BooleanSupplier](https://docs.oracle.com/javase/8/docs/api/java/util/function/BooleanSupplier.html): Supplier of boolean-valued results.  
[IntSupplier](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntSupplier.html):Supplier of int-valued results.  
[LongSupplier](https://docs.oracle.com/javase/8/docs/api/java/util/function/LongSupplier.html): Supplier of long-valued results.  
[DoubleSupplier](https://docs.oracle.com/javase/8/docs/api/java/util/function/DoubleSupplier.html): Supplier of double-valued results.

**Develop code that uses binary versions of functional interfaces.**

**Ans-** The following functional interfaces:

* Predicate<T>
* Consumer<T>
* Function<T,R>
* UnaryOperator<T>

Represent an operation that takes one argument. But there are versions of these interfaces that take two arguments called. These are the binary versions.

The binary versions of the functional interfaces are:

**BiPredicate<L, R>**  
(No primitive versions)

**BiConsumer<T, U>**  
[**ObjIntConsumer<T>**](https://docs.oracle.com/javase/8/docs/api/java/util/function/ObjIntConsumer.html)**:** Operation that accepts an Object-valued and an int-valued argument and returns no result.  
[**ObjLongConsumer<T>**](https://docs.oracle.com/javase/8/docs/api/java/util/function/ObjLongConsumer.html)**:**Operation that accepts an Object-valued and a long-valued argument and returns no result.  
[**ObjDoubleConsumer<T>**](https://docs.oracle.com/javase/8/docs/api/java/util/function/ObjDoubleConsumer.html)**:** Operation that accepts an Object-valued and a double-valued argument and returns no result.

**BiFunction<T, U, R>**  
[**ToIntBiFunction<T, U>**](https://docs.oracle.com/javase/8/docs/api/java/util/function/ToIntBiFunction.html)**:** Function that accepts two arguments and produces an int-valued result.  
[**ToLongBiFunction<T, U>**](https://docs.oracle.com/javase/8/docs/api/java/util/function/ToLongBiFunction.html)**:**Function that accepts two arguments and produces a long-valued result.  
[**ToDoubleBiFunction<T, U>**](https://docs.oracle.com/javase/8/docs/api/java/util/function/ToDoubleBiFunction.html)**:** Function that accepts two arguments and produces a double-valued result.

**BinaryOperator<T>**  
[**IntBinaryOperator**](https://docs.oracle.com/javase/8/docs/api/java/util/function/IntBinaryOperator.html)**:** Function operation upon two int-valued operands and producing an int-valued result.  
[**LongBinaryOperator**](https://docs.oracle.com/javase/8/docs/api/java/util/function/LongBinaryOperator.html)**:** Function operation upon two long-valued operands and producing a long-valued result.  
[**DoubleBinaryOperator**](https://docs.oracle.com/javase/8/docs/api/java/util/function/DoubleBinaryOperator.html)**:**Function operation upon two double-valued operands and producing a double-valued result.

**Develop code that uses the UnaryOperator interface.**

**Ans-** The function of the UnaryOperator interface is to take an object, do something with it and then return an object of the same type. The UnaryOperator is used to work on a single operand and it returns the same type as an operand. UnaryOperator can be used as lambda expression to pass as an argument.

**Eg-**

**import java.util.function.UnaryOperator;**

**// Create class that implements the UnaryOperator interface**

**public class UnaryOperatorExample implements UnaryOperator<String>{**

**public String apply(String text) {**

**return text+".txt";**

**}**

**}**

**class UnaryOperatorTest {**

**public static void main(String args[]){**

**UnaryOperatorExample uoe = new UnaryOperatorExample();**

**String text = "lambda-tutorial";**

**String newText = uoe.apply(text);**

**System.out.println(newText);**

**}**

**}**

When the class is executed, the result is the text string lambda-tutorial.txt written to the console.