Java Functional Interface and Lamda Implementation

The term *Java functional interface* was introduced in Java 8

A functional interface in Java is an interface that contains only a single abstract (unimplemented) method.

A functional interface may have defaulted, and static methods may have an implementation, in addition to the single unimplemented method.

`@FunctionalInterface` Annotation is added so that we can mark an interface as a functional interface. Even without it, your interface will be treated as functional as long as it has just one abstract method. **This ensures that the interface can't have more than one abstract method.**

Here, it has only one abstract method, so that we can omit the annotation`@FunctionalInterface.`

Important Points/Observations for functional interface:

1. A functional interface has only one abstract method, but it can have multiple default methods.
2. `@FunctionalInterface` Annotation is used to ensure an interface can't have more than one abstract method. The use of this annotation is optional.

3. The java.util.function package contains many built-in functional interfaces in Java 8.

Usages of Functional Interface

- The functional interface has been introduced in Java 8 to support the lambda expression in java 8. On the other hand, it can be said lambda expression is the instance of a functional interface.
- Java 8 Collections API has been rewritten, and a new Stream API is introduced that uses a lot of functional interfaces.
- Java 8 has defined a lot of functional interfaces in this package `java.util.function`. Some of the useful java 8 functional interfaces are `Consumer`, `Supplier`, `Function` and `Predicate`.

How to define Functional Interface

First, we can check a build-in functional interface. It is given below:

Here, The `Function` interface represents a function (method) that takes a single parameter (T) and returns a single value (R).

The `Function` interface actually contains a few extra methods in addition to the methods listed above. Still, since they all come with a default implementation, you do not have to implement these extra methods.

The only method you have to implement to implement the `Function` interface is the `apply()` method. Here is a `Function` implementation example:

Now the calling method is given below:

Here,

- The first example creates a new `AddSeedValueImplementation` instance and assigns it to a `Function` variable.
- Second, the example calls the `apply()` method on the `AddSeedValueImplementation` instance.
- Third, the example prints out the result (which is 9).

Now I am describing its lambda implementation:

Here,

- Firstly we are defining the calculations Function interfaces apply method. Here, n is supplied input, and (5L+n) is the sum of supplied input and hardcoded value 5
- Secondly, we are using the newly configured interface **Function**. Here in the apply method, we are passing the calculation.

Functional Interface with default method

We are just modifying the previous interface.

Here, we just added a default method named defaultMethodAdd.
Now calling mechanism is given below.

Functional Interface with two input and one result

Check the below code

Here,

- input is X and Y, and output is R

Now calling mechanism is

Here,

- We are defining function how it will behave. It is taking 2 parameters as input and multiplying and returning this result value.

Functional interface with string matching

Let's check below functional interface Predicate.

Here,

- Input is T, and it will be string, and output is R

Now calling mechanism is given below

Here,

- Creating a list of strings.
- Building predicate with logic string start with G
- Now extracting string starts with G into another list