Functional Interfaces In Java

A functional interface is an interface that contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. ***Runnable***, ***ActionListener***, ***Comparable*** are some of the examples of functional interfaces.
Before Java 8, we had to create anonymous inner class objects or implement these interfaces.

```
// Java program to demonstrate functional interface

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();
    }
}
```

Output:

```
New thread created
```

Java 8 onwards, we can assign lambda expression to its functional interface object like this:

```
// Java program to demonstrate Implementation of
// functional interface using lambda expressions

class Test
{
  public static void main(String args[])
  {

    // lambda expression to create the object
    new Thread(()->
       {System.out.println("New thread created");}).start();
  }
}
```

```
New thread created
```

@FunctionalInterface Annotation

@FunctionalInterface annotation is used to ensure that the functional interface can't have more than one abstract method. In case more than one abstract methods are present, the compiler flags an 'Unexpected @FunctionalInterface annotation' message. However, it is not mandatory to use this annotation.

```
// Java program to demonstrate lamda expressions to implement
// a user defined functional interface.

@FunctionalInterface
interface Square
{
    int calculate(int x);
}

class Test
{
    public static void main(String args[])
    {
        int a = 5;

        // lambda expression to define the calculate method
        Square s = (int x)->x*x;

        // parameter passed and return type must be
        // same as defined in the prototype
        int ans = s.calculate(a);
        System.out.println(ans);
    }
}
```

Output:

```
25
```

java.util.function Package:

The java.util.function package in Java 8 contains many builtin functional interfaces like-

- **Predicate:** The Predicate interface has an abstract method test which gives a Boolean value as a result for the specified argument. Its prototype is

```
public Predicate
{
   public boolean test(T  t);
 }
```

- **BinaryOperator:** The BinaryOperator interface has an abstract method apply which takes two argument and returns a result of same type. Its prototype is

```
public interface BinaryOperator
{
     public T apply(T x, T y);
}
```

- **Function:** The Function interface has an abstract method apply which takes argument of type T and returns a result of type R. Its prototype is

```
public interface Function
{
```

```
    public R apply(T t);
}
```

```
// A simple program to demonstrate the use
// of predicate interface
import java.util.*;
import java.util.function.Predicate;

class Test
{
    public static void main(String args[])
    {

        // create a list of strings
        List<String> names =
            Arrays.asList("Geek","GeeksQuiz","g1","QA","Geek2");

        // declare the predicate type as string and use
        // lambda expression to create object
        Predicate<String> p = (s)->s.startsWith("G");

        // Iterate through the list
        for (String st:names)
        {
            // call the test method
            if (p.test(st))
                System.out.println(st);
        }
    }
}
```

Output:

```
Geek
GeeksQuiz
Geek2
```

Important Points/Observations:

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 builtin functional interfaces in Java 8.

This article is contributed by **Akash Ojha** .If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Java.util.BitSet class methods in Java with Examples | Set 2

Shadowing of static functions in Java

How does default virtual behavior differ in C++ and Java ?

How are Java objects stored in memory?

How are parameters passed in Java?

Are static local variables allowed in Java?

final variables in Java

Default constructor in Java

Assigning values to static final variables in Java

Comparison of Exception Handling in C++ and Java

Does Java support goto?

Arrays in Java

Access specifier of methods in interfaces

Inheritance and constructors in Java

More restrictive access to a derived class method in Java

Article Tags : Java

Practice Tags : Java

9

To-do Done

4.1

Based on **17** vote(s)

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org