

Functional Interfaces

Prepared by github.com/all-mp

I

Introduction

Java has forever remained an **Object-Oriented Programming** language.

By object-oriented programming language, we can declare that **everything present in the Java programming language rotates throughout the objects, except** for some of the **primitive data types** and **primitive methods** for integrity and simplicity.

There are no **solely functions** present in a programming language called Java.

Functions in the Java programming language are part of a **class**, and if someone wants to use them, they have to **use the class or object of the class to call any function**.

II

What is Functional Interfaces ?

Functional Interface

An **Interface** that contains **exactly one abstract method** is known as **functional interface**.

It can have any number of default, static methods but can contain only one abstract method.

Functional Interface is also known as, **Single Abstract Method** Interfaces or **SAM** Interfaces.

Introduced in **Java 8** with **Lambda expressions** and **Method references** in order to make code more readable, clean, and straightforward.

Functional interfaces are used and executed by representing the interface with an annotation called **@FunctionalInterface**.

This feature in Java, which helps to achieve **functional programming approach**.

An informative **annotation** type used to indicate that an interface type declaration is intended to be a **functional interface** as defined by the Java Language Specification.

If an interface declares an abstract method overriding one of the public methods of **java.lang.Object**, that also does not count toward the interface's abstract method count since any implementation of the interface will have an implementation from **java.lang.Object** or elsewhere.

```
package me.praecher;

@FunctionalInterface // no usages
public interface Transform<T, R> {

    // this method transforms T into R
    R transformer(T t); // no usages
    // no problem
    boolean equals(Object obj);
    // no problem
    // no problem
    String toString();
}
```

```
package me.praecher;

@FunctionalInterface // no usages
public interface Transform<T, R> {

    // this method transforms T into R
    R transformer(T t); // no usages
    // problem
    // problem
    boolean equals(); // no usages
}
```

```
Transform<String, Integer> transform = new Transform<>() {
    @Override // no usages
    public Integer transformer(String s) {
        return s.length();
    }
};
```

Lambda Expression

```
Transform<String, Integer> transform = s -> s.length();
```

Method Reference

```
Transform<String, Integer> transform = String::length;
```

Constructor Reference

```
@FunctionalInterface
public interface Transform {

    String transformer();
}
```

```
Transform transform = String::new;
```

Note that instances of functional interfaces can be created with **lambda expressions**, **method references**, or **constructor references**.

If a type is annotated with this annotation type, compilers are required to generate an error message unless :

- ⋮ The type is an interface type and not an annotation type, enum, or class.
- ⋮ The annotated type satisfies the requirements of a functional interface.

However, the compiler will treat any interface meeting the definition of a functional interface as a functional interface regardless of whether or not a FunctionalInterface annotation is present on the interface declaration.

```
package java.lang;

import java.lang.annotation.*;

@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface FunctionalInterface {}
```

III

List of Functional Interfaces in Java (Introduction)

package java.util.function;

① **BiConsumer**
① **BiFunction**
① **BinaryOperator**
① **BiPredicate**
① **BooleanSupplier**
① **Consumer**
① **DoubleBinaryOperator**
① **DoubleConsumer**
① **DoubleFunction**
① **DoublePredicate**
① **DoubleSupplier**
① **DoubleToIntFunction**
① **DoubleToLongFunction**
① **DoubleUnaryOperator**
① **Function**
① **IntBinaryOperator**
① **IntConsumer**
① **IntFunction**
① **IntPredicate**
① **IntSupplier**
① **IntToDoubleFunction**
① **IntToLongFunction**
① **IntUnaryOperator**
① **LongBinaryOperator**
① **LongConsumer**
① **LongFunction**
① **LongPredicate**
① **LongSupplier**
① **LongToDoubleFunction**
① **LongToIntFunction**
① **LongUnaryOperator**
① **ObjDoubleConsumer**
① **ObjIntConsumer**
① **ObjLongConsumer**
① **Predicate**
① **Supplier**
① **ToDoubleBiFunction**
① **ToDoubleFunction**
① **ToIntBiFunction**
① **ToIntFunction**
① **ToLongBiFunction**
① **ToLongFunction**
① **UnaryOperator**

Predicate
Consumer
Function
Supplier
BiFunction
BinaryOperator

IV

List of Functional Interfaces in Java (explanation)