Generic Functional Interfaces in Java

1:45 Estimated 368 Words EN Language

Lokesh Gupta

February 26, 2022

Java Streams
Functional Interface, Generics, Java 8 Stream

Learn to create **generic functional interfaces with and without type restrictions** in Java 8 and later. Note that functional interfaces permit exactly one abstract method. These interfaces are also called **Single Abstract Method interfaces (SAM Interfaces)**.

1. Without Type Restrictions

1.1. Interface Definition

A functional interface can be defined that is generic for type `X` and has a functional method that accepts two arguments of type `X` and returns a value of type `X`.

```
@FunctionalInterface
public interface ArgumentsProcessor<X>
{
    X process(X arg1, X arg2);
}
```

This interface can be used for any type i.e. `ArgumentsProcessor<Integer>`, `ArgumentsProcessor<String>` or `ArgumentsProcessor<Employee>`.

1.2. Example

Java example to use generic functional interface with type `Integer`.

```
ArgumentsProcessor<Integer> multiplyProcessor = new ArgumentsProcessor<Integ
er>() {
    @Override
```

```
    public Integer process(Integer arg1, Integer arg2)
    {
        return arg1 * arg2;
    }
};

System.out.println(multiplyProcessor.process(2,3));     //6
```

Java example to use generic functional interface with type `String` .

```
ArgumentsProcessor<String> appendProcessor = new ArgumentsProcessor<String>
() {
    @Override
    public String process(String str1, String str2)
    {
        return str1  + " " + str2;
    }
};

System.out.println(appendProcessor.process("Hello", "World !!"));       //He
llo World !!
```

2. With Type Restrictions

2.1. Interface Definition

A functional interface can be defined that is **restricted to certain types** using `extends` keyword i.e. `X extends Number` .

```
@FunctionalInterface
public interface ArgumentsProcesso<X extends Number>
{
    X process(X arg1, X arg2);
}
```

This interface can be used for any type i.e. `ArgumentsProcessor<Integer>` , `ArgumentsProcessor<Double>` but not for `ArgumentsProcessor<String>` or `ArgumentsProcessor<Employee>` .

In the above example, the permitted type must extend the `Number` class.

2.2. Example

Java example to use generic functional interface with type `Integer` .

```
ArgumentsProcessor<Double> doubleMultiplier = new ArgumentsProcessor<Double>
() {
        @Override
        public Double process(Double arg1, Double arg2)
        {
            return arg1 * arg2;
        }
};

System.out.println(doubleMultiplier.process(4d, 6d));   //24.0
```

3. Specialized Functional Interfaces

Specialization is accomplished by extending or implementing the generic functional interface of one type. **The resulting interface or class is not generic for that type**.

```
@FunctionalInterface
public interface ArgumentsProcessor<Integer>
{
    Integer process(Integer arg1, Integer arg2);
}
```

```
ArgumentsProcessor<Integer> intMultiplier = (i1, i2) -> i1 * i2;

System.out.println(intMultiplier.process(4, 5));        //20
```

Drop me your questions related to **functional interfaces with generics**.

Happy Learning !!

> Sourcecode on Github

Further reading:

- Guide to Hibernate Criteria Queries
- Functional Interfaces in Java
- Java Concurrency Interview Questions
- Java Generics Tutorial
- Python Interview Questions and Answers
- TypeScript Generic Function, Class and Interface (+Examples)