



@techwithvishalraj



Java

Functional interface



Follow

Share

Functional Interfaces

- An Interface that contains exactly one abstract method is known as functional interface.
- Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces.
- It is a new feature in Java, which helps to achieve functional programming approach.
- It can contain any number of Object class methods.
- Java provides an annotation `@FunctionalInterface`, which is used to declare an interface as functional interface.

```
@FunctionalInterface
interface Welcome{
    void hello(); // abstract method
    // It can contain any number of Object class methods.
    int hashCode();
    String toString();
    boolean equals(Object object);
}

public class Test1 {
    public static void main(String[] args) {
        Welcome welcome = ()-> System.out.println("Hey hii :) ");
        welcome.hello();
        System.out.println(welcome.hashCode());
    }
}
```

output: Hey hii :)
455659002

Functional Interfaces

It can have any number of default, static methods.

```
@FunctionalInterface
interface MyFunctionalInterface {
    void abstractMethod();
    default void defaultMethod() {
        System.out.println("Default method implementation");
    }
    static void staticMethod() {
        System.out.println("Static method implementation");
    }
}

public class Test1 {
    public static void main(String[] args) {
        MyFunctionalInterface fi = ()->
            System.out.println("Abstarct method implementation");
        fi.abstractMethod();
        fi.defaultMethod();
        MyFunctionalInterface.staticMethod();
    }
}
```

output: Abstarct method implementation
Default method implementation
Static method implementation

Functional Interfaces

A functional interface can extend another interface only when it does not have any abstract method but it can contain default, static and object class methods.

```
interface MyInterface {  
    default void defaultMethod(){  
        System.out.println("Default method implementation");  
    }  
    int hashCode();  
}  
  
@FunctionalInterface  
interface MyFunctionalInterface extends MyInterface {  
    void abstractMethod();  
}  
  
public class Test3 {  
    public static void main(String[] args) {  
        MyFunctionalInterface fi = ()->  
            System.out.println("Abstarct method implementation");  
        fi.abstractMethod();  
        System.out.println(fi.hashCode());  
    }  
}
```

output: Abstarct method implementation
250421012

Predefined-Functional Interfaces

- Java provides predefined functional interfaces to deal with functional programming by using lambda and method references.
- Following is the list of some functional interface which are placed in java.util.function package.

| Interface | Description |
|-------------------|--|
| Consumer<T> | It represents an operation that accepts a single argument 'T' and does not returns result. |
| BiConsumer<T,U> | It represents an operation that accepts two input arguments and does not returns result. |
| Supplier<T> | It represents an operation that not take any arguments and returns a result of type 'T'. |
| Predicate<T> | represents operation that accepts a single argument 'T' and return boolean value. |
| Function<T,R> | It represents a function that accepts one argument 'T' and returns a result 'R'. |
| BiFunction<T,U,R> | It represents a function that accepts two arguments ('T', 'U) and returns a result 'R'. |

Consumer<T> Interface

It contains an abstract accept() method and a default andThen() method.

```
Consumer<String> con = (name) -> System.out.println("hello "+name);  
con.accept("vishal");           -----> hello vishal
```

BiConsumer<T,U> Interface

It contains an abstract accept() method and a default andThen() method.

```
BiConsumer<String, Integer> biConsumer = (name, age)->  
                                           System.out.println(name+ " : "+age);  
  
biConsumer.accept("vishal", 24);           -----> vishal : 24
```

Supplier<T> Interface

It has a single abstract method get().

```
Supplier<Double> supplier = ()-> Math.random()*100;  
System.out.println(supplier.get());       -----> 58.17961621550302
```

Predicate<T> Interface

it contains an abstract test() method and some default & static methods too.

```
Predicate<Integer> predicate = age-> age>18;  
Sop(predicate.test(12));
```

-----> false

Function<T,R> Interface

it contains an abstract apply() method and some default & static methods too.

```
Function<String, String> function = (name)-> "hello "+ name;  
Sop(function.apply("vishal"));
```

-----> hello vishal

BiFunction<T,U,R> Interface

it contains an abstract apply() method and default andThen() method.

```
BiFunction<String, String, String> biFunction = (s1, s2)-> s1+" "+s2;  
Sop(biFunction.apply("hello", "vishal"));
```

-----> hello vishal



@techwithvishalraj

*Thank
you!*



vishal-bramhankar



techwithvishalraj



Vishall0317

