

함수형 인터페이스



추상 메소드가 1개만 정의된 인터페이스

인터페이스 형태의 목적은 람다 표현식을 이용해 함수형 프로그래밍을 구현하기 위해서이다

! 자바에서 자료구조를 컬렉션 프레임워크로 미리 만들어 제공하듯이, 자주 사용할 것 같은 람다 함수 형태를 함수형 인터페이스 표준 API로 미리 만들어둔다

```
@FunctionalInterface
public interface Animal {
    public void method();
}
```

@FunctionalInterface 어노테이션을 붙여주면, 두 개 이상의 메소드 선언시 컴파일 오류가 발생시켜 개발자의 실수를 줄일 수 있음

표준 API 종류

```
import java.util.function.*;
```

함수적 인터페이스 종류로는 Consumer, Supplier, Function, Operator, Predicate가 있다

함수형 인터페이스	Descriptor	Method
Predicate	<code>T -> boolean</code>	<code>boolean test(T t)</code>
Consumer	<code>T -> void</code>	<code>void accept(T t)</code>
Supplier	<code>() -> T</code>	<code>T get()</code>
Function<T, R>	<code>T -> R</code>	<code>R apply(T t)</code>
Comparator	<code>(T, T) -> int</code>	<code>int compare(T o1, T o2)</code>
Runnable	<code>() -> void</code>	<code>void run()</code>
Callable	<code>() -> T</code>	<code>V call()</code>

Chapter 14. 람다와 스트림(Lambda & Stream)

Java의 정석
<http://www.codechobo.com>

14-7 java.util.function 패키지 - Quiz

Q. 아래의 빈 칸에 알맞은 함수형 인터페이스(java.util.function 패키지)를 적으시오.

①

f = () -> (int) (Math.random()*100)+1;

②

f = i -> System.out.print(i+", ");

③

f = i -> i%2==0;

④

f = i -> i/10*10;

함수형 인터페이스는 언제, 왜 사용할까?

함수형 인터페이스는 동작 파라미터화를 진행하는 것이 장점이다

즉 여러 동작이 존재할때 쓴다

1. 하나의 동작만 존재할때

자동차의 위치가 5이상인 자동차를 필터링하는 동작

```
public boolean filterCar(Car car, Predicate<Car> p) {
    return p.test(car);
}

public void carMove() {
    Car car = new Car(5);
    if(filterCar(car, carToPredicate -> carToPredicate.getPosition() >= 5)) {
        car.move();
    }
}

public void filterCar() {
    Car car = new Car(5);
    if(car.getPosition() >= 5) {
        car.remove();
    }
}
```

동작이 하나만 있으면 함수형 인터페이스를 사용하는 것보다, 직접 조건을 적어주는 것이 가독성이 좋음

함수형 인터페이스의 장점은 추상화로 인한 확장성과 재사용성이 있다

위의 예제에서, 다음 상황이 발생하면?

1. Car가 아니라, 다른 객체도 필터링 해야하는 상황
2. Car의 여러 조건을 필터링해야 하는 상황

→ 외부에서 사용하고 싶은 동작에 따라 구현체를 바꿔 끼울 수 있음

자바에서 기본으로 제공하는 함수형 인터페이스 사용 예제

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

필터1: 색상이 G, 위치가 5 이상인 자동차

필터2: 색상이 R, 위치가 3 이상인 자동차

필터3: 색상이 B, 위치가 7 이상인 자동차

```
public class Car {
    private final int position;
    private final String color;

    public Car(int position, String color) {
        this.position=position;
        this.color=color;
    }
    public int getPosition(){
        return position;
    }
    public String getColor() {
        return color;
    }
}
```

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

```
public class FilterCar {
    public List<Car> filter(List<Car> cars, Predicate<Car> filter)
```

```

        List<Car> result = new ArrayList<>();
        for(Car car: cars) {
            if(filterCondition.test(car)){
                result.add(car);
            }
        }
        return result;
    }
}

```

```

public class CarController {
    private final FilterCar filterCar;
    private final List<Car> cars;

    public CarController(FilterCar filterCar, List<Car> cars) {
        this.filterCar = filterCar;
        this.cars = cars;
    }

    public void getRank() {
        List<Car> filteredCars1 =
            filterCar.filter(cars, car->car.getColor());
        List<Car> filteredCars2 =
            filterCar.filter(cars, car->car.getColor());
        List<Car> filteredCars3 =
            filterCar.filter(cars, car->car.getColor());
    }
}

```

Functional Interface 만들기

```
@FunctionalInterface
interface CustomInterface<T> {
    T myCall();

    default void printDefault() {
        System.out.println("hello default");
    }
    static void printStatic() {
        System.out.println("hello static");
    }
}
```

사용

```
CustomInterface<String> customInterface = () -> "hello custom";

String s = customInterface.myCall();
customInterface.printDefault();
customInterface.printStatic();
```

2개의 인자를 받는 Bi 인터페이스

함수형 인터페이스	Descriptor	Method
BiPredicate	<code>(T, U) -> boolean</code>	<code>boolean test(T t, U u)</code>
BiConsumer	<code>(T, U) -> void</code>	<code>void accept(T t, U u)</code>
BiFunction	<code>(T, U) -> R</code>	<code>R apply(T t, U u)</code>

