람다란 함수를 의미하고 변수에 할당할 수 있다. 함수는 로직이다.

함수를 간략하면서도 명확한 식으로 표현할 수 있게 해준다.

람다란 한 마디로 코드블록이다.

기존의 코드 블록은 반드시 메서드 내에 존재해야 했다. 그래서 코드 블록만 갖고 싶어도 기존에는 코드 블록을 위해 메서드를, 다시 메서드를 사용하기 위해 익명 객체를 만들거나 하는 식이었다.

추상 메서드를 하나만 갖는 인터페이스를 자바 8부터는 함수형 인터페이스라고 한다. p.372

이런 함수형 인터페이스만을 람다식으로 변경할 수 있다.

함수형 인터페이스 = 변수에 저장 가능한 로직

함수형 인터페이스

지금까지 람다식이 메서드와 동등한 것처럼 설명해왔지만, 사실 람다식은 익명 클래스의 객체와 동등하다.

람다식으로 정의된 익명 객체의 메서드를 어떻게 호출할 수 있을 것인가?

🡪 참조변수가 있어야 객체가 있어야 한다.

타입 f = (int a, int b) -> a > b ? a : b;

// 참조변수의 f의 타입은 클래스 또는 인터페이스가 가능하다. 그리고 람다식과 동등한 메서드가 정의되어 있는 것이어야 한다.

Interface MyFunction { //인터페이스

Public abstract int max(int a, int b);

}

MyFunction f = new MyFunction() { //인터페이스를 익명 객체로 선언.

public int max(int a, int b){

return a > b ? a : b;

}

}

MyFunction f = (int a, int b) -> a > b ? a : b; // 익명 객체를 람다식으로 대체할 수 있다.

Int big = f.max(5,3) // 익명 객체의 메서드를 호출. (익명 객체, 람다 동일한 결과)

🡪이처럼 MyFunction 인터페이스를 구현한 익명 객체를 람다식으로 대체가 가능한 이유는 람다식도 실제로는 익명 객체이고, MyFunction인터페이스를 구현한 익명 객체의 메서드 max()와 람다식의 매개변수의 타입과 개수 그리고 반환값이 일치하기 때문이다.

🡪람다식을 참조변수로 다룰 수 있다는 것은 변수처럼 메서드를 주고받는 것이 가능해진 것이다.

(사실상 메서드가 아니라 객체를 주고받는 것이라 근복적으로 달라진 것은 아무것도 없다.)

🡪함수형 인터페이스에는 오직 하나의 추상 메서드만 정의되어 있어야 한다. 그래야 람다식과 인터페이스의 메서드가 1:1로 연결될 수 있기 때문이다.

매번 새로운 함수형 인터페이스를 정의하지 말고, 가능하면 이 패키지의 인터페이스를 활용하는 것이 좋다.

Java.lang.Runnable void run()

Java.util.function package

Interface Supplier<T> T get() (Returns: a result)

Interface Consumer<T> void accept(T t)

interface Function<T,R> R apply(T t)

interface Predicate<T> boolean test(T t)

기본형을 사용하는 함수형 인터페이스

Predicate<String> isEmptyStr = s -> s.length( ) == 0;

String s =””;

If(isEmptyStr.test(s)) // if(s.length() == 0)

System.***out***.println(“This is an empty String.”);

람다식

1, 람다식(Lambda Expression)은 간단히 말해서 메서드를 하나의 '식(expression)'으로 표현한 것이다.

람다식은 함수를 간략하면서도 명확한 식으로 표현할 수 있게 해준다. (메서드를 람다식으로 표현하면 메스드의 이름과 반환값이 없어지므로, 람다식을 '익명 함수(anonymous function)'이라고도 한다.)

나의 생각 정리) 람다식의 골격 : (매게 변수) 🡪 (함수 본문) 이렇게 표현한다.

2. 람다식 작성하기

메서드에서 1. 이름과 반환타입을 제거하고, 2. 매개변수 선언부와 몸통{ } 사이에 ->를 추가한다.

반환타입 메서드이름(매개변수 선언) {

문장들

}

~~반환타입 메서드이름~~(매개변수 선언) -> {

문장들

}

두 값 중에서 큰 값을 반환하는 메서드 max를 람다식으로 변환하면, 아래의 오른쪽과 같다.

Int man(int a, int b) {

Return a>b ? a : b;

}

(int a, int b) -> {

return a>b ? a : b;

}

반환값이 있는 메서드의 경우, return문 대신 ‘식(expression)’으로 대신 할 수 있다. 식의 연산결과가 자동적으로 반환값이 된다. 이때는 ‘문장(statement)’이 아닌 ‘식’이므로 끝에 ‘;’을 붙이지 않는다.

2. (int a, int b) -> { return a>b ? a : b; } --🡪 (int a, int b) -> a>b ? a : b

람다식에 선언된 매개변수의 타입은 추론이 가능한 경우는 생략할 수 있는데, 대부분의 경우에 생략 가능하다. 람다식에 반환타입이 없는 이유도 항상 추론이 가능하기 때문이다.

3. (int a, int b) -> a>b ? a : b --🡪 (a , b) -> a > b ? a : b

선언된 매개변수가 하나뿐인 경우에는 괄호( )를 생략할 수 있다. 단, 매개변수의 타입이 있으면 괄호( )를 생략할 수 없다.

(a) -> a \* a a -> a \* a //OK

(int a) -> a \* a int a -> a \* a //에러

마찬가지로 괄호{ } 안의 문장이 하나일 때는 괄호{ }를 생략할 수 있다. 이 때 문장의 끝에 ‘;’을 붙이지 않아야 한다는 것에 주의하자.

(String name, int i) -> {

System.***out***.println(name+”=”+i);

}

(String name, int i) -> System.***out***.println(name+”=”+i)

그러나 괄호{ } 안의 문장이 return문일 경우 괄호 { }를 생략할 수 없다.

(int a, int b) -> { return a>b ? a : b; } // OK

(int a, int b) -> return a>b ? a : b // Error