람다와 스트림

[람다]

함수형 프로그래밍

➡ 함수를 정의하고 이 함수를 데이터 처리부로 보내 데이터를 처리하는 기법

람다식

➡ 메서드 정의를 하나의 식으로 표현한 것

자바에서 함수적 프로그래밍 지원 기법으로서 익명 클래스 객체가 된다.

```
interface Sample {
   int calc(int n);
}
```

```
class SampleImpl implements Sample {
    public int calc(int n) {
        return n+1;
    }
}
Sample obj = new SampleImpl();
MyTest.pr(obj);
```

```
class MyTest {
    static void pr(Sample p) {
        p.calc(10);
    }
}
```

```
MyTest.pr(new Sample() {
    public int calc(int n) {
        return n+10;
    }
 });
```

```
MyTest.pr((int n) -> { return n+100; });
```

```
MyTest.pr((n) -> {return n+100;});
```

```
MyTest.pr(n -> {return n+100;});
```

```
MyTest.pr(n -> n+100);
```

- 함수형 인터페이스

인터페이스가 단 하나의 추상 메소드를 가지는 것

인터페이스

```
public interface Runnable {
  void run();
}
```

람다식

```
( ) -> { ··· }
```

인터페이스

```
@FunctionalInterface
public interface Calculable {
  void calculate(int x, int y);
}
```

람다식

```
( x, y ) -> { ··· }
```

인터페이스가 함수형 인터페이스임을 보장하기 위해서는 @FunctionalInterface 어노테이션을 붙인다. @FunctionalInterface : 컴파일 과정에서 추상 메소드가 하나인지 검사해 정확한 함수형 인터페이스를 작성할 수 있게 도와주는 역할이다.(함수형 인터페이스 체크 어노테이션)

- 매개변수가 없는 람다식

함수형 인터페이스의 추상 메소드에 매개변수가 없는 경우이다.
실행문이 두 개 이상일 경우에는 중괄호를 생략할 수 없고, 하나일 경우에만 생략할 수 있다.

```
( ) -> {
  실행문;
  실행문;
}
```

```
( ) -> 실행문
```

- 매개변수가 있는 람다식

함수형 인터페이스의 추상 메소드에 매개변수가 있는 경우이다.
매개변수를 선언할 때 타입은 생략할 수 있다.

```
(타입 매개변수, ··· ) -> {
  실행문;
  실행문;
}
```

```
(매개변수, ···) -> {
  실행문;
  실행문;
}
```

```
(타입 매개변수, ··· ) -> 실행문
```

```
(매개변수, ···) -> 실행문
```

매개변수가 하나일 경우에는 괄호를 생략할 수 있다.

```
매개변수 -> {
  실행문;
  실행문;
}
```

```
매개변수 -> 실행문
```

- 리턴값이 있는 람다식

함수형 인터페이스의 추상 메소드에 리턴 값이 있는 경우이다.
return 문 하나만 있을 경우에는 중괄호와 함께 return 키워드를 생략할 수 있다.
리턴값은 연산식 또는 리턴값이 있는 메소드의 호출식으로도 대체 가능하다.

```
(매개변수, ··· ) -> {
  실행문;
  return 값;
}
```

```
(매개변수, ···) -> return 값;
(매개변수, ···) -> 값
```

- 메소드 참조

메소드를 참조해 매개변수의 정보 및 리턴 타입을 알아내 람다식에서 불필요한 매개변수를 제거

```
(left, right) -> Math.max(left, right);
```

정적 메소드와 인스턴스 메소드 참조
람다식에서 정적 메소드를 참조할 경우 클래스 이름 뒤에 :: 기호를 붙이고 정적 메소드 이름을 기술

```
클래스 :: 메소드
```

인스턴스 메소드일 경우에는 객체를 생성한 다음 참조 변수 뒤에 :: 기호를 붙이고 인스턴스 메소드 이름을 기술

```
참조변수 :: 메소드
```

- 매개변수의 메소드 참조

람다식에서 제공되는 a 매개변수의 메소드를 호출할 때 b 매개변수를 매개 값으로 사용

```
(a, b) -> { a.instanceMethod(b); }
```

a의 클래스 이름 뒤에 :: 기호를 붙이고 메소드 이름을 기술

```
클래스 :: instanceMethod
```

String :: compareTolgnoreCase

- 생성자 참조

객체를 생성하는 것. 람다식이 단순히 객체를 생성하고 리턴하도록 구성되면 람다식을 생성자 참조로 대치 가능

```
(a, b) -> { return new 클래스(a, b); }
```

클래스 이름 뒤에 :: 기호를 붙이고 new 연산자를 기술

```
클래스 :: new
```

생성자가 오버로딩 되어 여러 개가 있을 경우, 컴파일러는 함수형 인터페이스의 추상 메소드와 동일한 매개변수 타입과 개수를 가지고 있는 생성자를 찾아 호출한다.
(해당 생성자가 존재하지 않으면 컴파일 오류 발생)