6주차 발표_민서연

# 스트림
# : 람다 표현식

01 스트림

# 스트림

자바8 API에 새로 추가된 기능.
데이터 처리 연산을 지원하도록 소스에서 추출된 연속된 요소

# 01 스트림

## 스트림

반복을 알아서 처리하고
결과 스트림 값을 어딘가 저장해주는 내부반복

## 컬렉션

사용자가 직접 요소를 반복하는
외부 반복

# 01 스트림

## 스트림

반복을 알아서 처리하고
결과 스트림 값을 어딘가 저장해주는 내부반복

## 컬렉션

사용자가 직접 요소를 반복하는
외부 반복

**스트림이 더 편하다!**

# 01 스트림

## 스트림

반복을 알아서 처리하고
결과 스트림 값을 어딘가 저장해주는 내부반복

## 컬렉션

사용자가 직접 요소를 반복하는
외부 반복

## 스트림이 더 편하다!

분할이 잘 이루어질 수 있는 데이터 구조이거나
연산작업이 독립적이면서
CPU 사용이 높은 작업에 적합!

# 02 스트림과 람다

## 람다식

함수를 간단한 식(Expression) 으로 표현한 것

메서드의 이름과 반환값이 생략된다는 점에서 익명함수라고도 불림

## 02 스트림과 람다

# 메서드 선언할 때!

## 기존 방식

모두 이름을 붙여주며 블럭으로 닫아주고,
return 도 해주고,
매개변수 타입도 정해줘야함

# 02 스트림과 람다

## 메서드 선언할 때!

### 기존 방식

모두 이름을 붙여주며 블럭으로 닫아주고,
return 도 해주고,
매개변수 타입도 정해줘야함

### 람다를 쓴다면?

이름과 타입 지정등의 과정을 생략 가능!
식(expression)임에도 불구하고
변수처럼 사용도 가능!

# 02 스트림과 람다

## 단점

람다에 대해 잘 모르는 사람에게는 가독성이 좋지는 않음

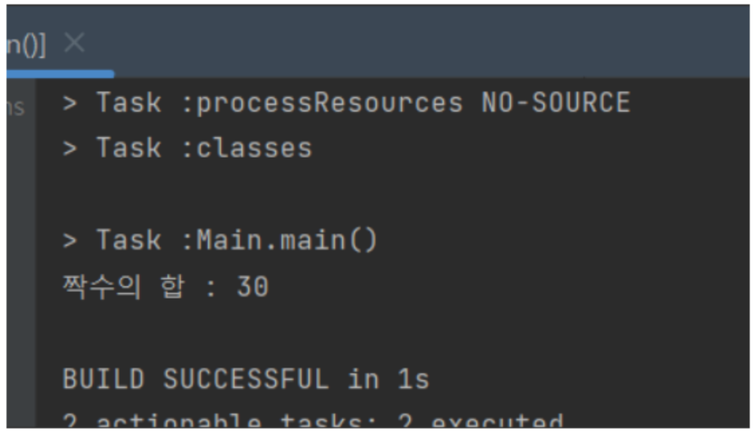
익명함수 기반이기 때문에 디버깅 부분이 어려움
-> 내부적으로 수행하는 작업이 더 많기 때문에 코드가 복잡해질수록
문제 발생 지점을 확인하기 힘들수 있다.

# 02 스트림과 람다

## 짝수를 찾아 합산하는 예시 - 반복문사용

```java
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        int sum = 0;
        for (int number : numbers) {
            if (number % 2 == 0) {
                sum += number;
            }
        }
        System.out.println("짝수의 합 : " + sum);
    }
}
```



# 02 스트림과 람다

## 짝수를 찾아 합산하는 예시 - 스트림과 람다식을 활용

```java
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        스트림과 람다식을 활용
        int sum = numbers.stream()
                .filter(number -> number % 2 == 0)
                .reduce( identity: 0, Integer::sum);

        System.out.println("짝수의 합 : " + sum);
    }
}
```

짝수만 필터링!

```
> Task :classes

> Task :Main.main()
짝수의 합 : 30

BUILD SUCCESSFUL in 1s
```

# 02 스트림과 람다

리스트의 문자열 요소 중 길이가 3 이하인 요소들을
대문자로 변환하여 출력 예시 - 반복문사용

```java
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("apple", "banana", "cat", "dog", "elephant");

        for (String word : words) {
            if (word.length() <= 3) {
                System.out.println(word.toUpperCase());
            }
        }
    }
}
```

```
> Task :processResources NO-SOURCE
> Task :classes

> Task :Main.main()
CAT
DOG
```

# 02 스트림과 람다

리스트의 문자열 요소 중 길이가 3 이하인 요소들을
대문자로 변환하여 출력 예시 - 스트림과 람다식 이용

```java
import java.util.Arrays;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("apple", "banana", "cat", "dog", "elephant");
        스트림과 람다식을 활용
        words.stream()
                .filter(word -> word.length() <= 3)
                .map(String::toUpperCase)
                .forEach(System.out::println);

    }
}
```

길이가 3이하인 문자열 필터링
대문자로 반환

```
> Task :processResources NO-SOURCE
> Task :classes

> Task :Main.main()
CAT
DOG

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed
```

# 03 추가 메서드 설명

## ■ filter 메서드

스트림 요소를 순회하면서 특정 조건을 만족하는 요소로 구성된 새로운 스트림을 반환하는 메서드이다.

- 특정 조건을 만족하는 요소로 구성된 새로운 스트림을 생성한다.
- 매개변수는 함수형 인터페이스인 Predicate 이므로, 람다 표현도 가능하다.
- 한개의 매개변수를 가지며 람다 함수는 Boolean 타입 값을 반환하는 람다 표현식 또는 Predicate 타입 객체가 전달 되어야함

Predicate
: 인수를 받아 boolean 값을 반환하는 함수형 인터페이스.

# 03 추가 메서드 설명

## ■ reduce 메서드

Stream의 요소들을 하나의 데이터로 만드는 작업을 수행하는 메서드이다.

- 정의된 연산이 아닌 프로그래머가 직접 구현한 연산을 적용한다.
- 최종 연산으로 스트림의 요소를 소모하며 연산을 수행한다.
- 두번째 요소로 전달되는 람다식에 따라 다양한 기능을 수행할 수 있다.
- 람다식을 직접 구현하거나 람다식이 긴 경우 BinaryOperator를 구현한 클래스를 이용한다.

# 03 추가 메서드 설명

■ map 메서드

Stream 내의 값들을 변환하여 새로운 Stream을 생성하는 메서드

- 반환하는 형태에 따라 여러 메서드가 존재한다.
- 일반 스트림과 기본형 특화 스트림 모두에서 제공하는 메서드 이다.
- 반환타입이 void로 보통 스트림의 요소를 출력하는 용도로 많이 사용한다.