자바 스트림 Stream

스트림의 흐름

1. **생성하기** : 스트림 인스턴스 생성
2. **가공하기** : 필터링 및 맵핑 등 원하는 결과를 만들어가는 중간 작업
3. **결과 만들기** : 최종적으로 결과를 만들어내는 작업

```
전체 -> 맵핑 -> 필터링 1 -> 필터링 2 -> 결과 만들기 -> 결과물
```

일반 스트림과 특화 스트림

일반스트림?

Stream<Integer>, Stream<String> 과 같은 것들

특화 스트림?

자바 스트림 API는 숫자 스트림을 효율적으로 처리할 수 있도록 세 가지 기본형 특화 스트림을 제공한다.

IntStream, DoubleStream, LongStream

- 기본형 종류

종류	데이터형	크기(byte / bit)
논리형	boolean	1 / 8
문자형	char	2 / 16
정수형	byte	1 / 8

종류	데이터형	크기(byte / bit)
정수형	short	2 / 16
정수형	int	4 / 32
정수형	long	8 / 64
실수형	float	4 / 32
실수형	double	8 / 64

이러한 특화 스트림은 숫자 스트림의 합계를 계산하는 sum, 최댓값 요소를 검색하는 max 같이 자주 사용하는 숫자 관련 리듀싱 연산 수행 메서드를 제공한다!

특화 스트림으로 변환

`mapToInt`, `mapToDouble`, `mapToLong` 메서드를 이용하면 일반 스트림을 특화 스트림으로 변환할 수 있다.

```
int totalPayPrice = pays.stream()
    .mapToInt(Pay::getPayPrice)
    .sum();
```

저 `::` 은 도대체 뭘까?
자바의 **메소드 레퍼런스**라는 것으로, 람다식을 다른 방식으로 표현한 것이다.
예
람다식 `(text)-> System.out.println(text)`
메소드 레퍼런스 `System.out :: println`

메소드 레퍼런스는 `[ClassName] :: [MethodName]` 형식으로 입력하고,
메소드를 호출하는 것이지만 괄호()는 써주지 않고 생략한다!

`.mapToInt(Pay::getPayPrice)`
람다식로 변환하면?
`.mapToInt(onePay → onePay.getPayPrice)`

List<Integer>를 int[]로 변환하고 싶다면?

`mapToInt` 이용!

```
int[] arr = list.stream()
                .mapToInt(i -> i) // == mapToInt(Integer::intValue)
                .toArray();
```

boxed 메서드

boxed() 메서드는 IntStream 같이 원시 타입에 대한 스트림 지원을 클래스 타입(예: `IntStream` -> `Stream<Integer>`) 으로 전환해준다.

사용 예: int 자체로는 Collection에 못 담기 때문에 Integer 클래스로 변환하여 `List<Integer>` 로 담기 위해 사용한다.

```
List<Integer> list1 = Arrays.stream(arr) // == IntStream.of(arr)
                            .boxed()
                            .collect(Collectors.toList());
```

유용하고 중요한 메서드! (주관적)

연산	형식	반환 형식	사용된 함수형 인터페이스 형식	함수 디스크립터
filter	중간 연산	Stream<T>	Predicate<T>	T -> boolean
distinct	중간 연산	Stream<T>		
skip	중간 연산	Stream<T>	Long	
limit	중간 연산	Stream<T>	Long	
map	중간 연산	Stream<R>	Function<T, R>	T -> R
flatMat	중간 연산	Stream<R>	Function<T, Stream<R>>	T -> Stream<R>
sorted	중간 연산	Stream<T>	Comparator<T>	(T, T) -> int
anyMatch	최종 연산	boolean	Predicate<T>	T -> Boolean
noneMatch	최종 연산	Boolean	Predicate<T>	T -> Boolean
allMatch	최종 연산	Boolean	Predicate<T>	T -> Boolean
findAny	최종 연산	Optional<T>		
findFirst	최종 연산	Optional<T>		
forEach	최종 연산	Void	Consumer<T>	T -> void
collect	최종 연산	R	Collector<T, A, R>	
reduce	최종 연산	Optional<T>	BinaryOperator<T>	(T, T) -> T
count	최종 연산	long		

filter

if문이라고 생각하면 편하다.

일치하는 모든 요소를 포함하는 스트림으로 **반환**한다.

메뉴 List에서 베지테리안 메뉴만 찾아서 리스트로 저장하고 싶다면?

```
List<Dish> vegetarianMenu = menu.stream()
                                .filter(Dish::isVegetarian)
                        // 리턴값이 true면 다음 단계 진행, false면 버려짐!
                                .collect(Collectors.toList());
```

collect

스트림의 요소들을 우리가 원하는 자료형으로 변환할 수 있다.

`toList()`, `toSet()`, `toMap()`, `toCollection()`

```
stream.collect(Collectors.toSet()); //set으로 변환
stream.collect(Collectors.toList()); //list 변환
stream.collect(Collectors.joining()); //요소를 다 이어붙여서 하나의 string으로 변환
stream.collect(Collectors.joining(", ")); //요소들 사이에 ","을 넣어서 한개의 string 반환
```

List나 Set이 아닌 특정 컬렉션을 지정하려면 `toCollection()` 에 해당 컬렉션의 생성자 참조를 매개변수로 넣어주면 된다.

```
ArrayList<String> list = names.stream()
                              .collect(Collectors.toCollection(ArrayList::new));
```

- collect()는 아니지만, 배열로 변환하고 싶다면 `stream.toArray()` 사용!

```
Person[] personArray1 = personStream1.toArray(Person[]::new);
```

map

스트림을 우리가 원하는 모양의 새로운 스트림으로 변환할 수 있다.

스트림 내 문자열을 모두 대문자로 변경한 스트림 만들기?

```
Stream<String> stream = names.stream()
                             .map(String::toUpperCase);
```

~~Match

- 하나라도 조건을 만족하는 요소가 있는지 *`anyMatch`*
- 모두 조건을 만족하는지 *`allMatch`*
- 모두 조건을 만족하지 않는지 *`noneMatch`*

```
List<String> names = Arrays.asList("Eric", "Elena", "Java");

boolean anyMatch = names.stream()
  .anyMatch(name -> name.contains("a"));
boolean allMatch = names.stream()
  .allMatch(name -> name.length() > 3);
boolean noneMatch = names.stream()
  .noneMatch(name -> name.endsWith("s"));
```

결과는 모두 **true!**

스트림 활용 예제

1️⃣ int[] numbers = {1, 2, 3, 4, 5};
위와 같은 배열이 있을 때,
스트림을 활용해서 아래와 같은 동작을 수행하자!

1. 홀수만 골라서,
2. 2를 곱한 뒤,
3. 배열에 저장한다.

```
int[] result = Arrays.stream(data)
                     .filter(num -> num % 2 == 1)
                     .map(num -> num * 2)
                     .toArray();
```

2️⃣ int[] data = {5, 6, 4, 2, 3, 1, 1, 2, 2, 4, 8};
위와 같은 배열이 있을 때,
스트림을 활용해서 아래와 같은 동작을 수행하자!

1. 짝수만 뽑은 뒤,
2. 중복을 제거하고,
3. 역순으로 정렬해서
4. 출력한다.

```
Arrays.stream(data)
      .boxed() // IntStream에서 -> Stream<Integer>로 변환
      .filter(n -> n % 2 == 0)
      .distinct() // 중복 제거
      .sorted(Comparator.reverseOrder())
      .forEach(System.out :: println); //  == .forEach((num) -> System.out.println(num))
```

- `boxed` 하는 이유?
  - `sorted(Comparator.reverseOrder())`를 하기 위함.
    IntStream이 제공하는 `sorted()`는 내림차순이 불가능!!

3️⃣ 프로그래머스 음양 더하기

```
class Solution {
    public int solution(int[] absolutes, boolean[] signs) {
        int sum = 0;
        for(int i=0; i<absolutes.length; i++) {
            if(!signs[i]) {
                absolutes[i] *= -1;
            }
            sum +=  absolutes[i];
        }
        return sum;
    }
}
```

스트림 활용 리팩토링

```
import java.util.stream.IntStream;
class Solution {
    public int solution(int[] absolutes, boolean[] signs) {
        return IntStream.range(0, absolutes.length) // {0,1,2, ... absolutes.length-1}
                        .map(i -> !signs[i] ? -absolutes[i] : absolutes[i])
                        .sum();
    }
}
```