
이펙티브 자바

item45 ~ item46

24/04/11

Item 45

스트림은 주의해서 사용하라

아이템 45 스트림은 주의해서 사용하라

스트림이란?

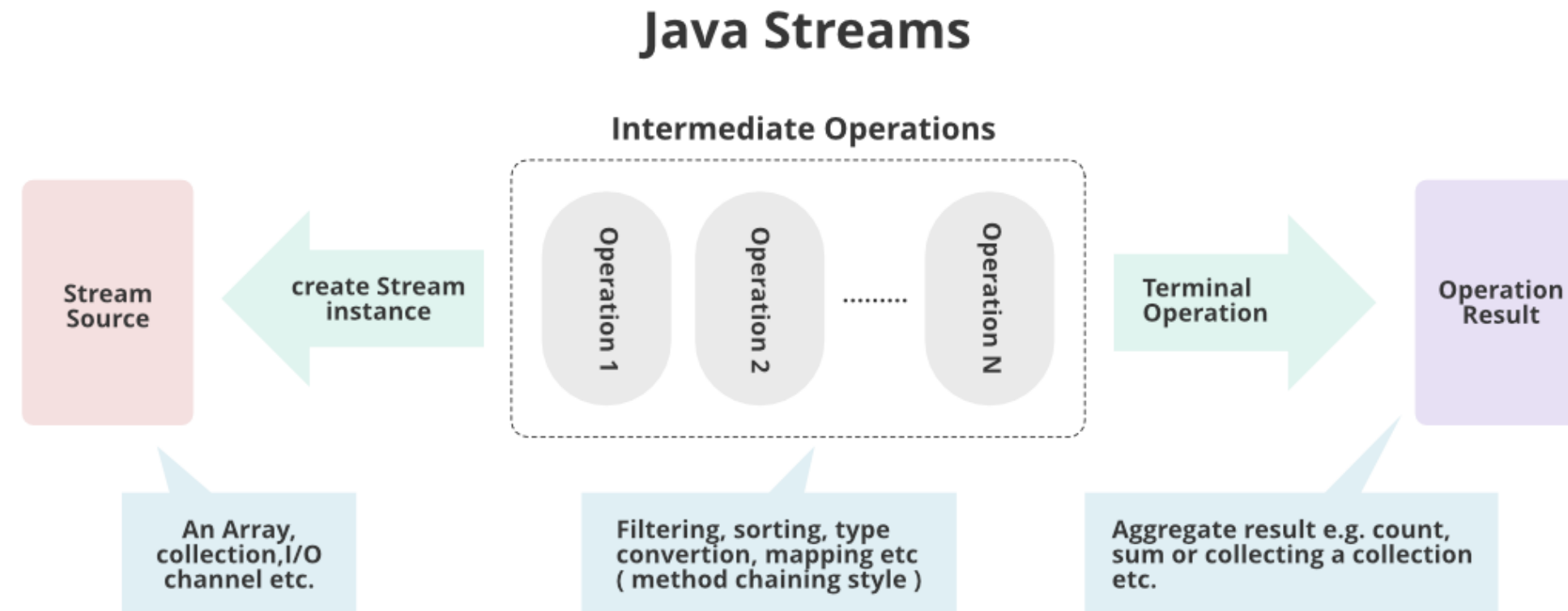
스트림 API는 다량의 데이터 처리 작업을 위해 자바8부터 추가.

이 API의 핵심 개념은 다음 두가지이다.

- 1.스트림(stream) : 데이터 원소의 유한 혹은 무한 시퀀스
- 2.스트림 파이프라인(stream pipeline) : 이 원소들로 수행하는 연산 단계를 표현

대표적으로 컬렉션, 배열, 파일, 정규표현식 매처, 난수 생성기, 다른 스트림 등등이 있다.

아이템 45 스트림은 주의해서 사용하라



스트림 파이프라인 연산은 소스 스트림에서 출발해 종단 연산에서 끝이 난다.
그 사이에는 하나 이상의 중간 연산(intermediate operation)이 존재한다.

중간 연산 : 스트림을 어떠한 방식으로 변환하는 역할을 해 메서드 체이닝을 만든다.
ex) filter(), map()

종단 연산 : 종단 연산은 마지막 중간 연산이 내놓은 스트림에 최종 연산을 가한다.
ex) forEach(), match(), collect() 등

아이템 45 스트림은 주의해서 사용하라

NOTE

스트림 파이프라인은 lazy evaluation된다.

평가는 종단 연산이 호출될 때 이루어지므로, 결과값이 나올 때까지 계산을 늦추게 된다.

이로 인해 실제로 필요하지 않은 중간 데이터를 탐색하는 시간을 없애서 속도를 향상시킨다.

→ short-circuit이라고도 부름.



하지만 종단 연산이 없다면 스트림은 멈추지 않으므로, 종단 연산을 빼먹는 일은 절대 없도록 한다.



아이템 45 스트림은 주의해서 사용하라

스트림 예시

```
<code />

List<Integer> transactionsIds =
    transactions.stream()
        .filter(t → t.getType() == Transaction.GROCERY)
        .sorted(comparing(Transaction::getValue).reversed())
        .map(Transaction::getId)
        .collect(toList());
```

가독성

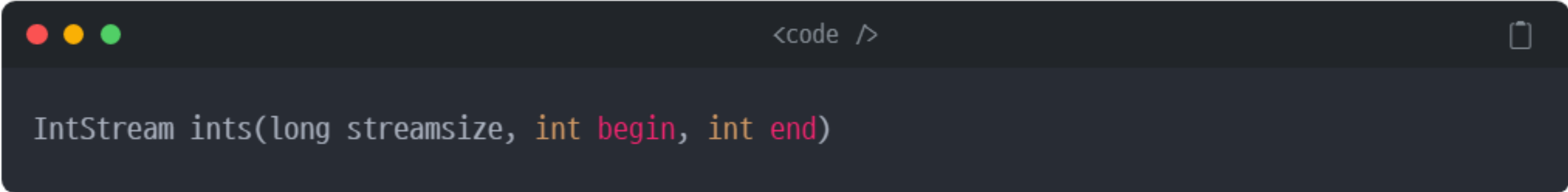


시간



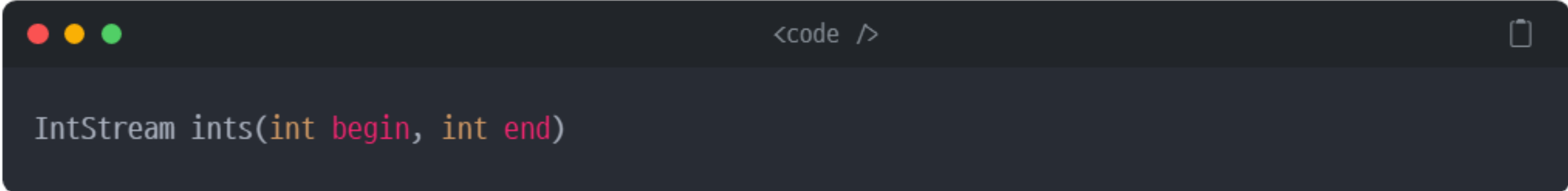
아이템 45 스트림은 주의해서 사용하라

유한 스트림 vs 무한 스트림



```
IntStream ints(long streamsize, int begin, int end)
```

유한 스트림 : 생성할 때 크기가 정해져 있는 스트림

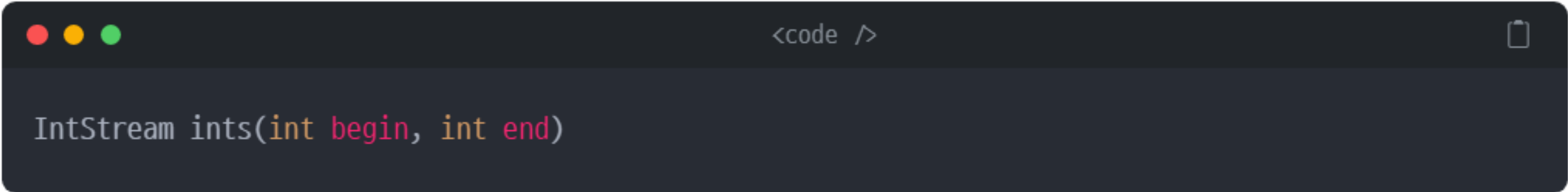


```
IntStream ints(int begin, int end)
```

무한 스트림 : 크기가 정해지지 않은 스트림

아이템 45 스트림은 주의해서 사용하라

무한 스트림



```
IntStream ints(int begin, int end)
```

무한 스트림은 short-circuit 연산을 통해 유한 스트림으로 변환 ○

중복을 제거하는 `distinct()` 혹은 전체 데이터를 정렬하는 `sort()` 연산들을 stateful 연산이라 한다.

그러나 이러한 연산들은 lazy evaluation을 무효화하고 결과를 생성하기 전에 전체 데이터를 탐색하게 한다.

아이템 45 스트림은 주의해서 사용하라

```
<code />

void streamLazySortTest(){
    List<String> noList = List.of("abcde" , "adceb", "dcabeda", "gssss");
    noList.stream()
        .filter(x → x.length() ≥ 5)
        .peek(x → System.out.println("중간연산: " + x))
        .sorted()
        .limit(2)
        .forEach(x → System.out.println("종단연산: " + x))
}
```

리소스 

아이템 45 스트림은 주의해서 사용하라

순차성

기본적으로 스트림 파이프라인은 순차 수행.
파이프라인을 병렬로 실행하려면? ➡ **parallel** 메서드를 호출

아이템 45 스트림은 주의해서 사용하라

 언제 스트림을 활용?

코드 가독성과 유지보수 측면을 항상 고려.

따라서 스트림과 반복문을 적절히 조합해서 사용하자.

➡ 스트림을 사용하도록 리팩토링하되, 새 코드가 나아보일때만 반영!

아이템 45 스트림은 주의해서 사용하라

스트림 활용방안

스트림에서 할 수 없는 일

- 지역 변수를 수정하는 것, 람다는 사실상 final인 변수만 읽을 수 있다.
- 람다로는 return, break, continue 등 특정 조건에서 반복문을 종료하거나 건너뛰는 것이 불가능하다.
 - 또한 메서드 선언에 명시된 예외도 던질 수 없다.

스트림에서 하면 좋은 일

- 원소들의 시퀀스를 일관되게 변환 map()
- 원소들의 시퀀스를 필터링 : filter()
- 원소들의 시퀀스를 하나의 연산을 사용해 결합
- 원소들의 시퀀스를 컬렉션에 모으기 : collect()
- 원소들의 시퀀스 중 특정 조건을 만족하는 원소 찾기 : filter()

아이템 45 스트림은 주의해서 사용하라

결론

NOTE

결론

반복문과 스트림 방식중 이해하고 유지보수하기에 더 편한 방식을 사용하자. 여기에는 본인의 주관과 동료들의 의견을 반영하는 것이 베스트이다.

수많은 작업은 둘을 조합했을 때 가장 멋지게 해결되므로, 둘 다 익혀두고 항상 비교해가며 사용할 것.

Item 46

스트림에서는 부작용 없는 함수를 사용하라

아이템 46 스트림에서는 부작용 없는 함수를 사용하라

어떤 작업을 스트림 파이프라인으로 표현해야 장점이 있을까?

➡스트림 패러다임의 핵심은 계산을 일련의 변환으로 재구성 하는 부분.

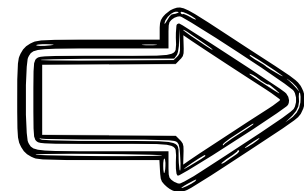
각 변환 단계는 **순수 함수**여야 한다.

➡순수 함수란, 다른 가변 상태에 영향을 받지 않고 오직 입력만이 결과에 영향을 미치는 함수를 의미

아이템 46 스트림에서는 부작용 없는 함수를 사용하라

```
<code />

Map<String, Long> freq = new HashMap<>();
try (Stream<String> words = new Scanner(file).tokens()) {
    words.forEach(word → {
        freq.merge(word.toLowerCase(), 1L, Long::sum);
    });
}
```



```
<code />

Map<String, Long> freq;
try (Stream<String> words = new Scanner(file).tokens()) {
    freq = words.collect(groupingBy(String::toLowerCase, counting()));
}
```



forEach 연산은 종단 연산 중 기능이 가장 적고 덜 스트리스럽다. (병렬화 불가능하기에)
forEach 연산은 스트림 계산 결과를 보고할 때만 사용하자.

아이템 46 스트림에서는 부작용 없는 함수를 사용하라

스트림을 올바르게 안전하게 사용하기 위해서는 수집기를 사용한다.

수집기는 ***축소** 전략을 캡슐화한 블랙박스 객체이다.

➡축소란 스트림의 원소들을 객체 하나에 취합하는 것을 의미.

아이템 46 스트림에서는 부작용 없는 함수를 사용하라

<https://docs.oracle.com/javase/10/docs/api/java/util/stream/Collectors.html>

↑ 위 공식 문서에서 Collectors의 다양한 메서드 확인 가능

OVERVIEWMODULEPACKAGECLASSUSETREEDEPRECATEDINDEXHELP

PREV CLASSNEXT CLASSFRAMESNO FRAMESALL CLASSESSEARCH:

SUMMARY: NESTED | FIELD | CONSTR | METHODDETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.util.stream
Class Collectors

java.lang.Object
java.util.stream.Collectors

public final class Collectors
extends Object

Implementations of `Collector` that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.

The following are examples of using the predefined collectors to perform common mutable reduction tasks:

```
// Accumulate names into a List
List<String> list = people.stream()
    .map(Person::getName)
    .collect(Collectors.toList());

// Accumulate names into a TreeSet
Set<String> set = people.stream()
    .map(Person::getName)
    .collect(Collectors.toCollection(TreeSet::new));

// Convert elements to strings and concatenate them, separated by commas
String joined = things.stream()
    .map(Object::toString)
    .collect(Collectors.joining(", "));
```

아이템 46 스트림에서는 부작용 없는 함수를 사용하라

NOTE

결론

스트림 파이프라인을 만들 때는 부작용이 없는 함수 객체를 사용하고, `forEach`는 반드시 계산 결과를 보고할 때만 이용하자.