Effective Java 읽기 스터디 20mins for weekly review

6주차 2021.04.05

Chapter 7

발표자: 김정수

목차

1. 7장 간단 리뷰

Lambda

람다는 코드를 간결하게 해준다.
메서드 참조를 쓰면 람다보다 코드를 더 간결하게 해준다.

Stream

스트림 API는 코드를 간결하게 해준다.
너무 간결하면 가독성이 떨어진다.

2. Item 42: 익명 클래스 vs 람다

익명 클래스

: 추상 메서드가 하나인 인터페이스를 구현한 클래스로, 함수 객체를 만들 때 쓰인다.

- 익명 클래스의 인스턴스에 접근할 수 있다.
- this 키워드는 익명클래스의 인스턴스 자신을 가리킨다.

-> 함수 객체가 자신을 참조해야 한다면 익명클래스를 쓰자.

람다식

: 추상 메서드가 하나인 인터페이스를 구현한 인스턴스.

- 자기 인스턴스에 접근할 수 없다.
- this 키워드는 바깥 인스턴스를 가리킨다.

-> 자신의 인스턴스에 접근할 일이 없으면 코드를 간략하게 하기 위해 람다식을 쓰자.

3. Item 43: 람다 vs 메서드 참조

람다식

: 추상 메서드가 하나인 인터페이스를 구현한 인스턴스.

- 매개변수의 이름이 의미를 가지고 있을 경우 유용함.
- 메서드와 람다가 같은 클래스에 있을 때 코드를 좀 더 줄일 수 있다.

메서드 참조

: 람다와 기능이 같은 메서드의 참조를 전달하여 메서드를 호출하는 방식이다.

- 람다보다 더 간결한 코드를 짤 수 있다.
- 람다로 작성할 코드를 새로운 메서드에 담고, 메서드 참조를 사용하면 된다.
- 제네릭 함수 타입을 구현할 수 있다.(람다에서는 불가능)

-> 일부 상황을 제외하고, 메서드 참조가 더 간결한 코드를 짤 수 있게 한다.

3. Item 43: 람다 vs 메서드 참조

메서드 참조

: 람다와 기능이 같은 메서드의 참조를 전달하여 메서드를 호출하는 방식이다.

- 람다보다 더 간결한 코드를 짤 수 있다.
- 람다로 작성할 코드를 새로운 메서드에 담고, 메서드 참조를 사용하면 된다.
- 제네릭 함수 타입을 구현할 수 있다.(람다에서는 불가능)

-> 일부 상황을 제외하고, 메서드 참조가 더 간결한 코드를 짤 수 있게 한다.

참조 유형

1. 정적 메서드를 가리키는 메서드 참조
2. 한정적 인스턴스 메서드 참조: 참조 대상을 특정한다. (인수 일치)
3. 비한정적 인스턴스 메서드 참조: 참조 대상을 특정하지 않는다. (스트림에 쓰인다)
4. 클래스 생성자를 가리키는 메서드 참조
5. 배열 생성자를 가리키는 메서드 참조

메서드 참조 유형	Ex
정적 메서드	Integer::parseInt
한정적 인스턴스	Instant.now()::isAfter
비한정적 인스턴스	String::toLowerCase
클래스 생성자	TreeMap<K,V>::new
배열 생성자	int[]::new

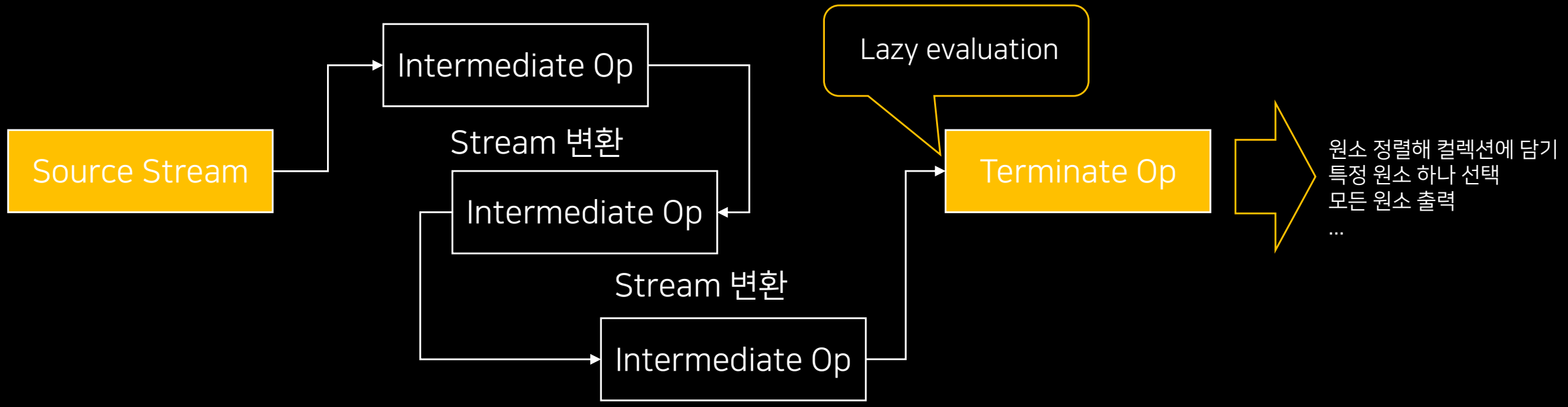
4. Item 44: 표준 함수형 인터페이스

표준 함수형 인터페이스
: 자바 라이브러리에 정의된 함수형 인터페이스. java.util.function 패키지에 있다.

- 대부분 primitive type만 지원한다. (성능을 위해 Boxing된 타입 사용 X)

함수형 인터페이스를 직접 구현할 때 주의할 점
- Comparator 인터페이스처럼 자주 쓰이며, 이름 자체가 용도를 명확히 설명해 줘야 한다.
- 항상 @FunctionalInterface 애너테이션을 붙이자.
- Overloading은 하지 말자.

스트림
: Java8에서 추가된 API로, 계산을 일련의 transformation으로 재구성하여 다량의 데이터 처리 작업을 돕는다.
각 변환 단계는 입력만이 결과에 영향을 주는 순수 함수여야 한다.



4. Item 45: 스트림

java.util.stream.Collectors
: collector는 스트림에서 계산을 수행한 후 원소를 모으는 역할을 한다.

toList()
: 리스트 컬렉션 타입을 반환한다.

toSet()
: 집합 컬렉션 타입을 반환한다.

toCollection(CollectionFactory)
: 지정한 컬렉션 타입을 반환한다.

groupingBy(..)
: 다중정의되어 있으며, 입력으로 분류 함수를 받고 collector를 출력한다. 다운스트림 인수를 추가해서 반환할 collector를 지정해 줄 수 있다.

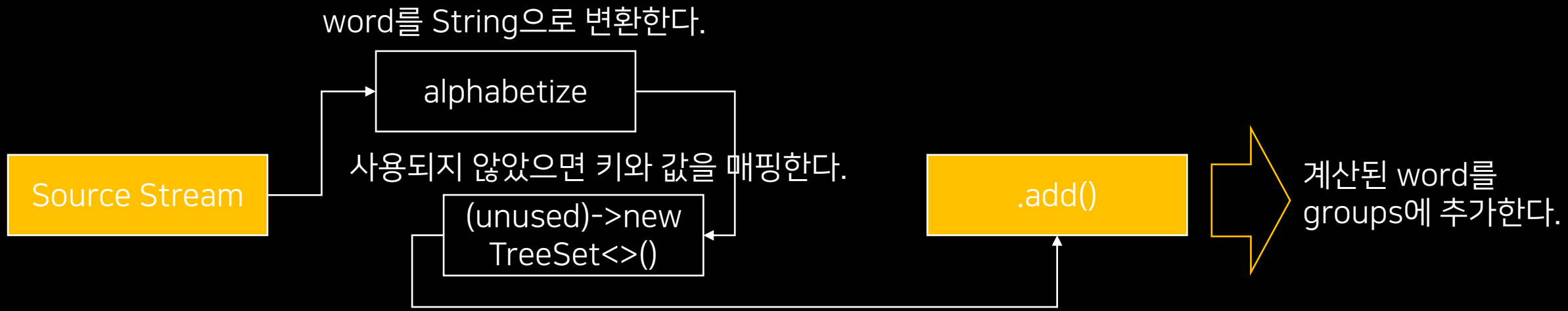
toMap(keyMapper, valueMapper)
: 함수 두 개를 인수로 받아 그 결과를 각각 key value로 받는다.

toMap(keyMapper, valueMapper, Comparator)
: key에 해당되는 value 중 비교자에 따라 우선되는 value가 매핑된다.

toMap(keyMapper, valueMapper,
(oldVal, newVal)-> newVal)
: key 충돌이 나면 마지막 값을 취한다.

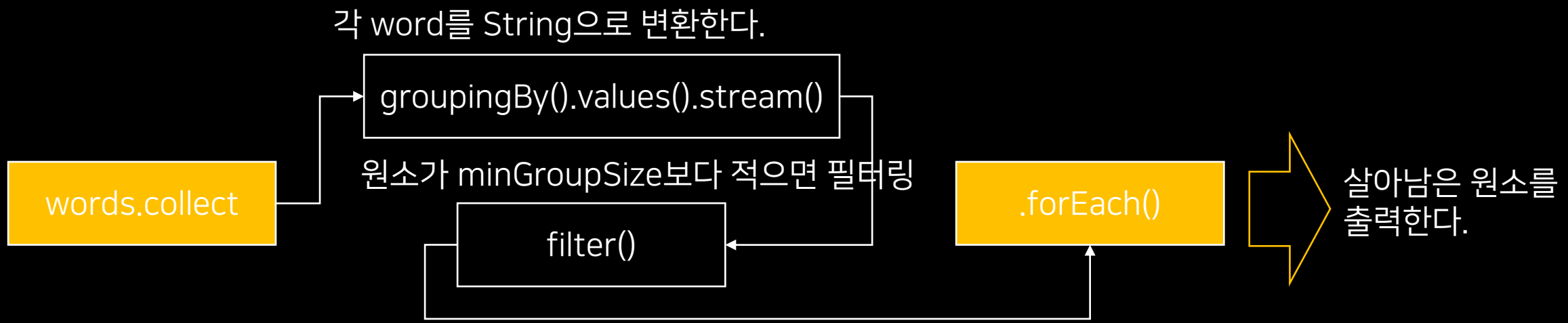
joining()
: CharSequence 인스턴스 스트림에만 적용된다. 인수에 따라 delimiter, suffix, prefix를 붙인다.

```
groups.computeIfAbsent(alphabetize(word),
        (unused) -> new TreeSet<>()).add(word);
```



4. Item 45: 스트림

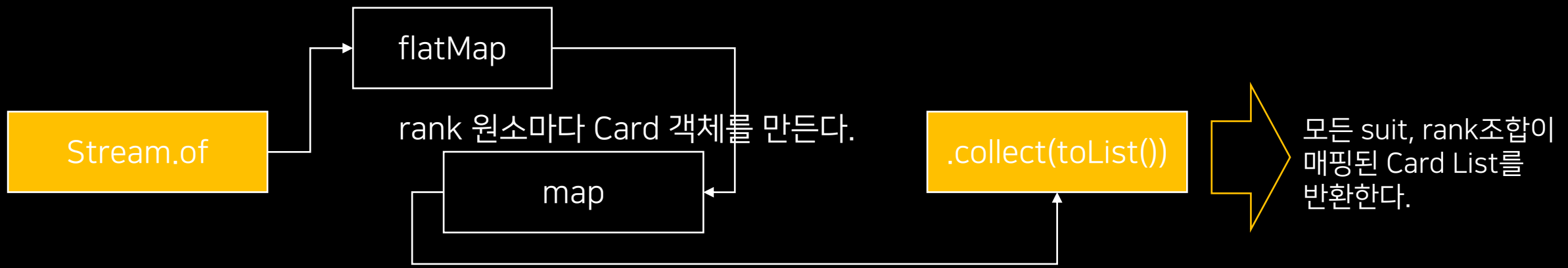
```
words.collect(groupingBy(word->alphabetize(word)))
.values().stream()
.filter(group -> group.size() >= minGroupSize)
.forEach(g -> System.out.println(g.size() + ": " + g));
```



4. Item 45: 스트림

```
Stream.of(Suit.values())
        .flatMap(suit ->
                Stream.of(Rank.values())
                        .map(rank -> new Card(suit, rank)))
.collect(toList());
```

suit 원소에 대해 다음 stream의 원소들을 매핑한다.



4. Item 45: 스트림

기존 코드는 스트림을 사용하도록 리팩터링하되, 새 코드가 더 나아 보일 때만 반영하자.
과한 스트림은 코드 가독성과 유지보수성을 해친다.
스트림 병렬화는 성능 최적화 수단이기 때문에 충분한 테스트를 통해 성능이 향상되는지 검증해야 한다.

스트림 사용하지 말아야 할 때

- 지역변수 수정
- break, continue 문 사용할 때
- 예외 throw할 때
- 반환 타입으로는 컬렉션이 낫다.

스트림이 적합할 때

- 원소들의 시퀀스를 변환하고,
- 필터링하고,
- 하나의 연산을 사용해 결합할 때.
- 특정 조건을 만족하는 원소를 찾을 때

Q&A