[Item46] 스트림에서는 부작용 없는 함수를 사용하라

요약

- 스트림 객체와 관련된 객체에 함수 객체가 건네지게 되는데
- 이 때 건네지는 모든 함수 객체는 부작용이 없는 순수함수여야 한다
- forEach() 는 스트림 계산 결과를 보고할 때만 사용하고, 계산하는 데는 쓰지 말자.
- toList(), toSet(), toMap(), groupingBy(), joinging() 과 같은 수집기 팩터리를 잘 익혀두자

응용

- 스트림을 활용해 어떤 단어가 몇번 나오는지 빈도표를 초기화하고, 빈도표에서 가장 흔한 단어 10개를 뽑아낼 수 있다
- 어떤 Enum 이 있을 때 맵으로 매핑하기 (문자열을 열거 타입 상수에 매핑)
- 다양한 음악가의 앨범들을 담은 스트림을 가지고, 음악가와 그 음악가의 베스트 앨범을 연관 짓기

스트림 패러다임

- 스트림은 그저 또 하나의 API 가 아닌 함수형 프로그래밍에 기초한 패러다임이다
- 스트림이 제공하는 표현력, 속도, 병렬성을 얻으려면 API 는 말할 것도 없고 이 패러다임까지 함께 받아들여야 한다
- 스트림 패러다임의 핵심은 계산을 일련의 변환으로 재구성 하는 부분이다.
- 이 때 각 변환 단계는 가능한 이전 단계의 결과를 받아 처리하는 순수 함수여야 한다

순수함수란

- 오직 입력만이 결과에 영향을 주는 함수.
- 다른 가변 상태를 참조하기 않고, 함수 스스로도 다른 상태를 변경하지 않는다.
- 이렇게 하려면 스트림 연산에 건네는 함수 객체는 모두 부작용이 없어야 한다

잘못된 코드!

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

위의 코드는 스트림 코드라고 할 수 없다! 스트림 코드를 가장한 반복적 코드이다

- 길고
- 읽기 어렵고
- 유지보수에도 좋지 않다

문제점?

- 외부 상태(freq) 를 수정하는 람다를 실행하기 때문

ForEach()

- 스트림이 수행한 연산 결과를 보여주는 일 이상을 하면 좋지 않은 코드이다
- forEach 연산은 스트림 계산 결과를 보고할 때만 사용하고, 계산하는 데는 쓰지 말자.

제대로 수정하면

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

수집기(collector)

- java.util.stream.Collectors
- 스트림의 원소들을 객체 하나에 취합한다
- 일반적으로 컬렉션 객체를 생성한다
  - toList(), toSet(), toMap()
  - 프로그래머가 지정한 컬렉션 반환 - toCollection(collectionFactory)

Collectors 메서드들

- 대부분 스트림을 맵으로 취합하는 기능을 함
- Collectors.toMap()
- BinaryOperator.maxBy
- groupingBy()
- toCollection(collectionFactory)
- groupingBy()
- groupingByConcurrent()
- partitioningBy()
- summing()
- averaging()
- summarizing()
- reducing()
- filtering()
- mapping()
- flatMapping()
- collectingAndThen()

예제

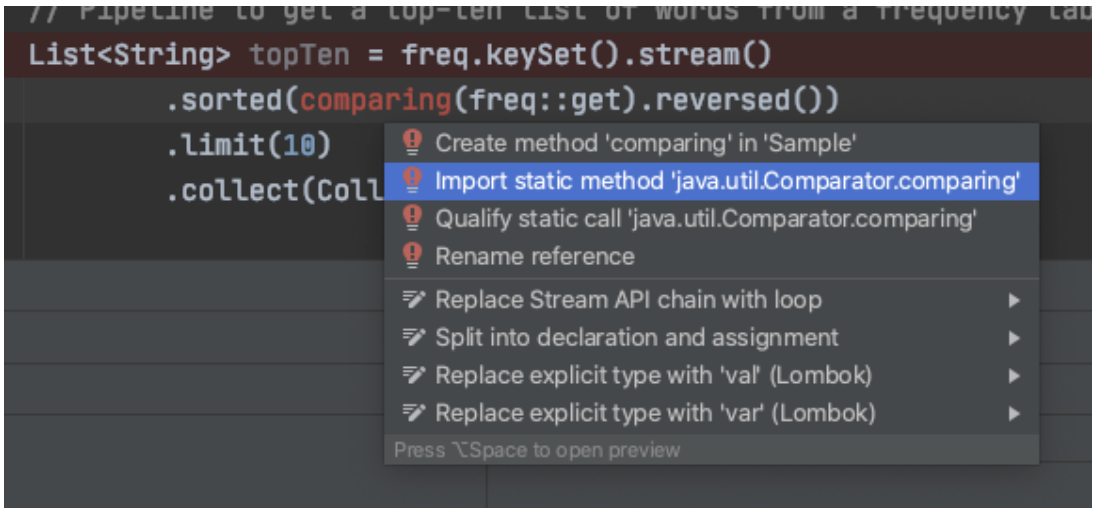
▼ 1. 빈도표에서 가장 흔한 단어 10개를 뽑아내는 스트림 파이프라인

```
Map<String, Long> freq = new HashMap<>();
for(Integer i=0; i<100; ++i) {
    freq.put(i.toString(), Long.valueOf(i));
}

List<String> topTen = freq.keySet().stream()
  .sorted(comparing(freq::get).reversed())
  .limit(10)
  .collect(toList());
  // .collect(Collectors.toList());
```

- toList() 정적 임포트

  여기서 toList() 는 Collectors 의 static 메소드이다.

  Collectors.toList() 로 사용해도 되지만 정적 임포트 하여 쓰면



  스트림 파이프라인 가독성이 좋아져 흔히 이렇게 사용한다

- `.sorted(comparing(freq::get).reversed())`

```
// Sream.java
Stream<T> sorted(Comparator<? super T> comparator);

// Comparator.java
```

```
public static <T, U extends Comparable<? super U>> Comparator<T> comparing(
  Function<? super T, ? extends U> keyExtractor)

default Comparator<T> reversed() {
    return Collections.reverseOrder(this);
}

// Map.java
V get(Object key);
```

comparing(freq::get) → Comparator 객체

`Comparator.comparing()` 인데 가독성을 높이기 위해 정적 임포트하여 사용

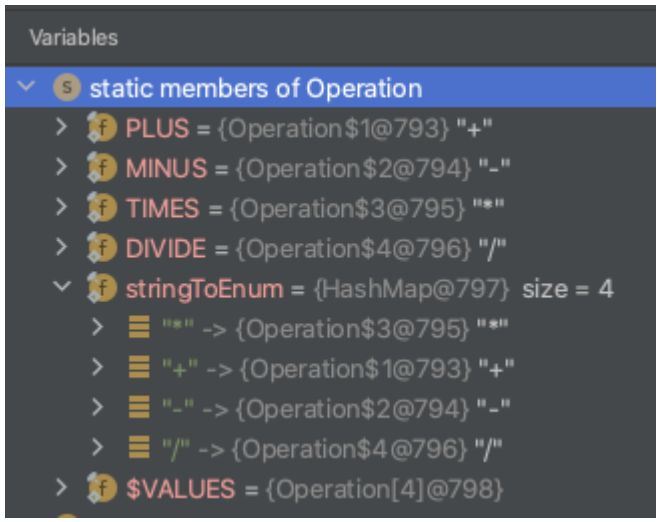
키 추출 함수를 인수로 받아 그 키를 기준으로 순서를 정하는 비교자를 반환하는 함수

- freq::get

  comparing() 의 키 추출 함수 인자로 사용됨

  한정적 메서드 참조방식

▼ 2. toMap() 수집기를 이용, 문자열을 열거 타입 상수에 매핑 - 키 매퍼, 값 매퍼




```
public enum Operation {
    PLUS("+") {
        public double apply(double x, double y) { return x + y; }
    },
```

```
    MINUS("-") {
        public double apply(double x, double y) { return x - y; }
    },
    TIMES("*") {
        public double apply(double x, double y) { return x * y; }
    },
    DIVIDE("/") {
        public double apply(double x, double y) { return x / y; }
    };

    private final String symbol;

    Operation(String symbol) { this.symbol = symbol; }

    @Override public String toString() { return symbol; }

    public abstract double apply(double x, double y);

    // Implementing a fromString method on an enum type (Page 164)
    private static final Map<String, Operation> stringToEnum =
            Stream.of(values()).collect(
                    toMap(Object::toString, e -> e)
            );
}
```

```
public static <T, K, U>
Collector<T, ?, Map<K,U>> toMap(Function<? super T, ? extends K> keyMapper,
                                Function<? super T, ? extends U> valueMapper) {
    return new CollectorImpl<>(HashMap::new,
                               uniqKeysMapAccumulator(keyMapper, valueMapper),
                               uniqKeysMapMerger(),
                               CH_ID);
}
```

```
Stream.of(values()).collect(
        toMap(Object::toString, e -> e)
);

Stream.of(values()).collect(
        toMap(e -> e.toString(), e -> e)
);
```

만약 스트림 원소 다수가 같은 키를 사용한다면 파이프라인은 `IllegalStateException` 을 던지며 종료된다.

▼ 3. toMap() 수집기 이용, 다양한 음악가의 앨범들을 담은 스트림을 가지고, 음악가와 그 음악가의 베스트 앨범을 연관 짓기

```
class Artist {
    String name;

    public Artist(String name) {
        this.name = name;
    }
}
```

```
class Album {
    String name;
    Artist artist;
    int sales;

    public Album(String name, Artist artist, int sales) {
        this.name = name;
        this.artist = artist;
        this.sales = sales;
    }

    public Artist getArtist() {
        return artist;
    }

    public int getSales() {
        return sales;
    }
}
```

```
public class TopHitsSample {

    public static void main(String[] args) {

        Set<Album> albums = new HashSet<>();

        Artist IU = new Artist("IU");
        Artist ZionT = new Artist("ZionT");

        albums.add(new Album("1집", IU, 1000));
        albums.add(new Album("2집", IU, 2000));
        albums.add(new Album("3집", IU, 3000));

        albums.add(new Album("1집", ZionT, 1000));
        albums.add(new Album("2집", ZionT, 2000));
```

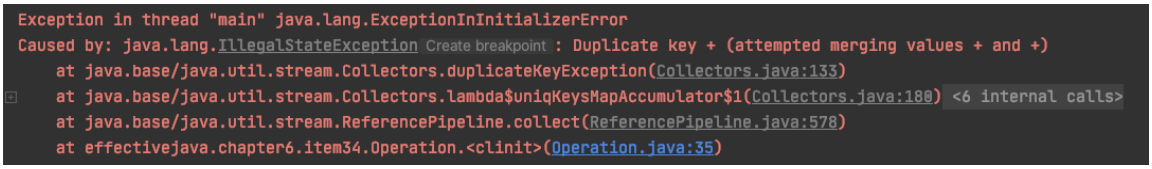
```
        Map<Artist, Album> topHits = albums.stream().collect(
            toMap(Album::getArtist, a->a, maxBy(comparing(Album::getSales)))
        );

    }

}
```

인수 3개를 받는 toMap() 을 사용하면 어떤 키와 그 키에 연관된 원소들 중 하나를 골라 연관 짓는 맵을 만들 때 유용하다

`maxBy()` 는 `Comparator<T>` 를 입력받아 `BinaryOperator<T>` 를 돌려준다

▼ 4. toMap() 수집기를 이용, 문자열을 열거 타입 상수에 매핑하려는데 +가 여러개로 키가 중복된다면?

```
public enum Operation {
    PLUS("+") {
        public double apply(double x, double y) { return x + y; }
    },
    MINUS("-") {
        public double apply(double x, double y) { return x - y; }
    },
    TIMES("*") {
        public double apply(double x, double y) { return x * y; }
    },
    DIVIDE("/") {
        public double apply(double x, double y) { return x / y; }
    }
    PLUS2("+") {
        public double apply(double x, double y) { return x / y; }
    };

    private final String symbol;

    Operation(String symbol) { this.symbol = symbol; }

    @Override public String toString() { return symbol; }

    public abstract double apply(double x, double y);

    // Implementing a fromString method on an enum type (Page 164)
    private static final Map<String, Operation> stringToEnum =
            Stream.of(values()).collect(
                    toMap(Object::toString, e -> e)
            );
}
```



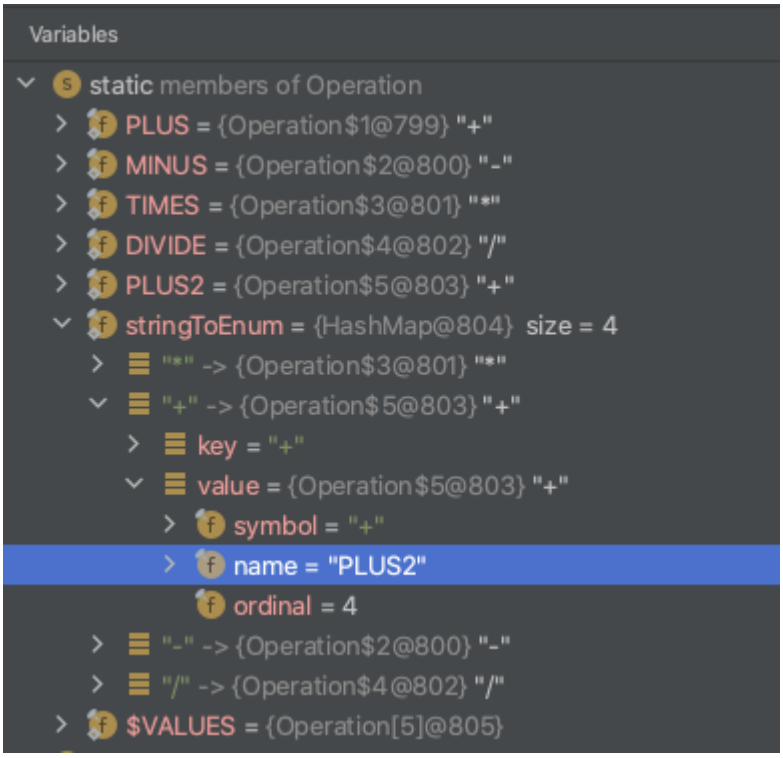
이럴 때 toMap() 3번째 인자를 사용하면

키 하나에 연결해준 값들이 여러개인 경우, 마지막에 쓴 값을 취하는 수집기를 만들 수 있다.

```
Stream.of(values()).collect(
        toMap(Object::toString, e -> e)
);
```

위의 코드를 아래 코드로 수정하면 된다

```
Stream.of(values()).collect(
        toMap(Object::toString, e -> e, (oldVal, newVal) -> newVal)
);
```


Variables
static members of Operation
PLUS = {Operation$1@799} "+"
MINUS = {Operation$2@800} "-"
TIMES = {Operation$3@801} "*"
DIVIDE = {Operation$4@802} "/"
PLUS2 = {Operation$5@803} "+"
stringToEnum = {HashMap@804} size = 4
"*" -> {Operation$3@801} "*"
"+" -> {Operation$5@803} "+"
key = "+"
value = {Operation$5@803} "+"
symbol = "+"
name = "PLUS2"
ordinal = 4
"-" -> {Operation$2@800} "-"
"/" -> {Operation$4@802} "/"
$VALUES = {Operation[5]@805}