JAVA Stream

> Java 8버전 이상부터는 Stream API를 지원한다

자바에서도 8버전 이상부터 람다를 사용한 함수형 프로그래밍이 가능해졌다.

기존에 존재하던 Collection과 Stream은 무슨 차이가 있을까? 바로 **'데이터 계산 시점'**이다.

Collection

- 모든 값을 메모리에 저장하는 자료구조다. 따라서 Collection에 추가하기 전에 미리 계산이 완료되어있어야 한다.
- 외부 반복을 통해 사용자가 직접 반복 작업을 거쳐 요소를 가져올 수 있다(for-each)

Stream

- 요청할 때만 요소를 계산한다. 내부 반복을 사용하므로, 추출 요소만 선언해주면 알아서 반복 처리를 진행한다.
- 스트림에 요소를 따로 추가 혹은 제거하는 작업은 불가능하다.

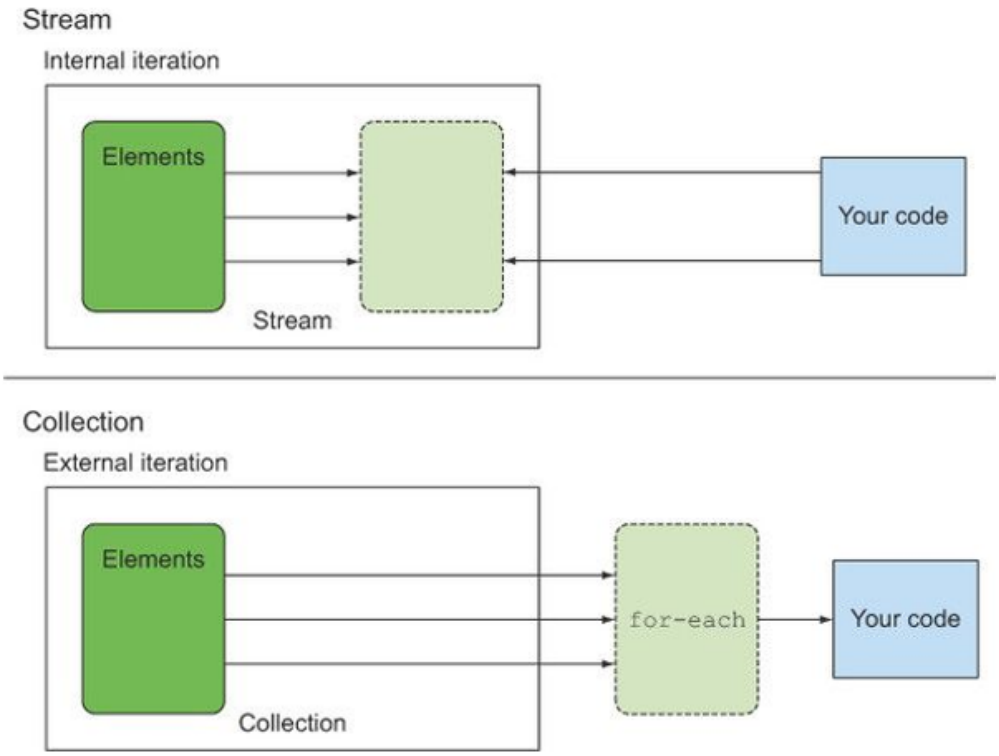
> Collection은 핸드폰에 음악 파일을 미리 저장하여 재생하는 플레이어라면, Stream은 필요할 때 검색해서 듣는 멜론과 같은 음악 어플이라고 생각하면 된다.

외부 반복 & 내부 반복

Collection은 외부 반복, Stream은 내부 반복이라고 했다. 두 차이를 알아보자.

성능 면에서는 '내부 반복'이 비교적 좋다. 내부 반복은 작업을 병렬 처리하면서 최적화된 순서로 처리해준다. 하지만 외부 반복은 명시적으로 컬렉션 항목을 하나씩 가져와서 처리해야하기 때문에 최적화에 불리하다.

즉, Collection에서 병렬성을 이용하려면 직접 `synchronized`를 통해 관리해야만 한다.



Stream 연산

스트림은 연산 과정이 '중간'과 '최종'으로 나누어진다.

`filter, map, limit` 등 파이프라이닝이 가능한 연산을 중간 연산, `count, collect` 등 스트림을 닫는 연산을 최종 연산이라고 한다.

둘로 나누는 이유는, 중간 연산들은 스트림을 반환해야 하는데, 모두 한꺼번에 병합하여 연산을 처리한 다음 최종 연산에서 한꺼번에 처리하게 된다.

ex) Item 중에 가격이 1000 이상인 이름을 5개 선택한다.

```
List<String> items = item.stream()
                .filter(d->d.getPrices()>=1000)
                          .map(d->d.getName())
                          .limit(5)
                          .collect(tpList());
```

> filter와 map은 다른 연산이지만, 한 과정으로 병합된다.

만약 Collection 이었다면, 우선 가격이 1000 이상인 아이템을 찾은 다음, 이름만 따로 저장한 뒤 5개를 선택해야 한다. 연산 최적화는 물론, 가독성 면에서도 Stream이 더 좋다.

Stream 중간 연산

- filter(Predicate) : Predicate를 인자로 받아 true인 요소를 포함한 스트림 반환

- distinct() : 중복 필터링
- limit(n) : 주어진 사이즈 이하 크기를 갖는 스트림 반환
- skip(n) : 처음 요소 n개 제외한 스트림 반환
- map(Function) : 매핑 함수의 result로 구성된 스트림 반환
- flatMap() : 스트림의 콘텐츠로 매핑함. map과 달리 평면화된 스트림 반환

> 중간 연산은 모두 스트림을 반환한다.

Stream 최종 연산

- (boolean) allMatch(Predicate) : 모든 스트림 요소가 Predicate와 일치하는지 검사
- (boolean) anyMatch(Predicate) : 하나라도 일치하는 요소가 있는지 검사
- (boolean) noneMatch(Predicate) : 매치되는 요소가 없는지 검사
- (Optional) findAny() : 현재 스트림에서 임의의 요소 반환
- (Optional) findFirst() : 스트림의 첫번째 요소
- reduce() : 모든 스트림 요소를 처리해 값을 도출. 두 개의 인자를 가짐
- collect() : 스트림을 reduce하여 list, map, 정수 형식 컬렉션을 만듬
- (void) forEach() : 스트림 각 요소를 소비하며 람다 적용
- (Long) count : 스트림 요소 개수 반환

Optional 클래스

값의 존재나 여부를 표현하는 컨테이너 Class

- null로 인한 버그를 막을 수 있는 장점이 있다.
- isPresent() : Optional이 값을 포함할 때 True 반환

Stream 활용 예제

1. map()

```
List<String> names = Arrays.asList("Sehoon", "Songwoo", "Chan", "Youngsuk",
"Dajung");

names.stream()
    .map(name -> name.toUpperCase())
    .forEach(name -> System.out.println(name));
```

2. filter()

```
List<String> startsWithN = names.stream()
    .filter(name -> name.startsWith("S"))
    .collect(Collectors.toList());
```

3. reduce()

```
Stream<Integer> numbers = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
Optional<Integer> sum = numbers.reduce((x, y) -> x + y);
sum.ifPresent(s -> System.out.println("sum: " + s));
```

> sum : 55

4. collect()

```
System.out.println(names.stream()
                    .map(String::toUpperCase)
                    .collect(Collectors.joining(", ")));
```

[참고자료]

- 링크
- 링크