Stream API

1. Stream API

1-1. Stream API란?

- 컬렉션의 요소들을 람다식을 이용해 함수형 프로그래밍 방식으로 처리할 수 있게 해주는 API
- Stream API는 Java 8에서 도입됨

1-2. Stream API의 특징

- 원본 데이터를 변경하지 않음 (불변성)
- 일회용 (한 번 사용한 스트림은 재사용 불가, Iterator와 비슷)
- 지연 연산 (최종 연산이 수행되기 전까지 중간 연산은 실행되지 않음)
- 내부 반복을 통한 작업 수행
- 병렬 처리가 용이함

1-3. Stream 객체의 종류

인터페이스	설명
BaseStream	모든 스트림 인터페이스의 부모
Stream<T>	객체 요소를 처리하는 스트림
IntStream	int 요소를 처리하는 스트림
LongStream	long 요소를 처리하는 스트림
DoubleStream	double 요소를 처리하는 스트림

기본 타입 스트림 예시

```
// IntStream 사용 예시
IntStream intStream = IntStream.range(1, 5); // 1,2,3,4
LongStream longStream = LongStream.rangeClosed(1, 5); // 1,2,3,4,5
DoubleStream doubleStream = DoubleStream.of(1.1, 2.2, 3.3);
```

---

2. Stream 생성 방법

```
// 1. 배열로부터 스트림 생성
String[] arr = {"a", "b", "c"};
Stream<String> stream1 = Arrays.stream(arr);

// 2. 컬렉션으로부터 스트림 생성
List<String> list = Arrays.asList("a", "b", "c");
Stream<String> stream2 = list.stream();

// 3. Stream.of()를 사용한 생성
Stream<String> stream3 = Stream.of("a", "b", "c");
```

💡 **생성된 Stream 사용 시 주의사항**

- 스트림은 일회용이므로 한 번 사용한 스트림은 재사용할 수 없음
- 스트림 연산은 원본 데이터를 변경하지 않음
- 지연 연산으로 인해 최종 연산이 수행되기 전까지는 중간 연산이 실행되지 않음
- 병렬 처리 시 순서가 보장되지 않을 수 있음

---

3. Stream 연산의 종류

3-1. 중간 연산

- 스트림을 변환하는 연산으로, 결과로 새로운 스트림을 반환

연산	설명
filter()	주어진 조건에 맞는 요소만 선택
map()	각 요소를 새로운 형태로 변환
flatMap()	중첩된 구조를 단일 수준으로 평탄화
distinct()	중복된 요소 제거
sorted()	요소들을 정렬
peek()	스트림의 요소를 확인하며 디버깅
limit()	스트림의 요소 개수를 제한
skip()	스트림의 앞쪽 n개 요소를 건너뜀

중간 연산 예제

```
List<String> names = Arrays.asList("John", "Jane", "Jack");

// 메서드 체이닝을 통한 중간 연산 조합
names.stream()
    .filter(name → name.startsWith("J"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);
```

실행 결과:

```
JACK
JANE
JOHN
```

3-2. 최종 연산

- 스트림의 요소를 소모하여 결과를 만들어냄

연산	설명
forEach()	각 요소를 순회하며 지정된 작업 수행
collect()	스트림의 요소들을 컬렉션으로 변환
reduce()	스트림의 모든 요소를 하나의 값으로 결합
count()	스트림 요소의 총 개수 반환

min(), max()	스트림에서 최솟값과 최댓값 찾기
anyMatch(), allMatch(), noneMatch()	요소들의 조건 만족 여부 검사
findFirst(), findAny()	조건에 맞는 요소 찾기

---

4. Stream API 활용 예제

4-1. 학생 성적 처리 예제

```
class Student {
    private String name;
    private int score;

    // 생성자, getter, setter 생략
}

// 학생 목록 생성
List<Student> students = Arrays.asList(
    new Student("Kim", 80),
    new Student("Lee", 95),
    new Student("Park", 92)
);

// 90점 이상인 학생들의 이름을 정렬하여 출력
students.stream()
    .filter(s -> s.getScore() >= 90)
    .map(Student::getName)
    .sorted()
    .forEach(System.out::println);
```

4-2. 메서드 참조를 활용한 예제

```
List<String> words = Arrays.asList("hello", "world", "stream", "api");

// 일반 람다식
words.stream()
    .map(s -> s.toUpperCase())
    .forEach(s -> System.out.println(s));

// 메서드 참조 사용
words.stream()
    .map(String::toUpperCase)
    .forEach(System.out::println);
```