Stream

작성일시	@2023년 6월 19일 오후 2:35
종류	자바의정석 Chapter14
유형	
자료	
복습	☐
Spring Framwork	

💡 스트림이란?

우리는 많은 수를 다룰 때 컬렉션이나 배열에 데이터를 담고 원하는 결과를 얻기 위해 for문과 Iterator를 이용해서 코드를 작성해왔다.

-문제점
너무 길고 알아보기 어렵다, 재사용성도 떨어진다.
데이터 소스마다 다른 방식으로 사용해야 한다. (List → Collection.sort , 배열 →Arrays.sort())

- 문제를 해결하기 위한 Stream
데이터 소스를 추상화하고, 데이터를 다루는데 자주 사용되는 메서드들을 정의해 놨다.
데이터 소스가 무엇이던 간에 같은 방식으로 다룰 수 있게 만들어서 재사용성이 높아진다는 의미이다.

-특징
1. 데이터 소스를 변경하지 않는다.
데이터 소스로부터 데이터를 읽기만 할 뿐, 변경하지 않는다.

2. 일회용이다.
Iterator처럼 일회용이므로 한 번 사용한 스트림은 다시 사용할 수 없다.
필요시 다시 생성해야 한다.

3. 내부 반복으로 처리한다.
내부 반복이란 반복문을 메서드의 내부에 숨길 수 있다는 것을 의미한다.

4. 스트림의 연산
제공되는 다양한 연산을 통해 복잡한 작업들을 간단히 처리할 수 있다.

5. 중간연산과 최종 연산이 있다.
중간연산은 연속해서 사용이 가능하지만, 최종 연산은 마지막에 스트림을 소모하므로 한 번만 가능하다.

6. 지연된 연산
최종연산이 수행되기 전까지는 중간 연산이 수행되지 않는다.

▼ 중간연산 메서드

💡 Stream<T> distinct()
Stream<T> filter(Predicate<T> predicate)
Stream<T> limit(long maxSize)
Stream<T> skip(long n)
Stream<T> peek(Counsumer<T> action)
Stream<T> sorted(), Stream<T> sorted(Comparator<T> comparator)
Stream<R> map(Function<T,R> mapper)
DoubleStream, Int Stream, Long Stream → mapTo(Double,Int,Long),
flatMapTo(Int,Double,Long)

▼ 스트림 만들기(컬렉션)

💡 List
컬렉션의 최고 조상인 Collection에 stream()이 정의되어 있다.
그래서 자손인 List,Set을 구현한 컬렉션 클래스들은 모두 이 메서드로 스트림을 생성할 수 있다.

배열
배열을 소스로 하는 스트림을 생성하는 매서드는 Stream과 Arrays에 static메서드로 정의되어 있다.

```
/**
List로부터 스트림 생성
*/

List<Integer> list = Arrays.asList(1,2,3,4,5);
Stream<Integer> intStream = list.stream();

/*
배열로부터 스트림 생성
*/
Stream<T> Stream.of(T... values)
Stream<T> Stream.of(T[])
Stream<T> Arrays.stream(T[])
Stream<T> Arrays.stream(T[] array, int startInclusive, int EndExclusive)
```

스트림 만들기(정수)

```
IntStream.range(int begin, int end) end미포함
IntStream.rangeClosed(int begin,int end) end포함
intStream ints(begin, end)
LongStream longs("")
DoubleStream Doubles("")
```

▼ 람다식 - iterate(), generate()

💡 iterate()
씨앗값으로 지정된 값부터 시작ㅎ해서 , 람다식 f에 의해 계산 된 결과를 다시 seed값으로 해서 계산을 반복한다.
generate()
iterate()처럼 무한스트림을 생성하지만, 매개변수를 받지 않는 Supllier타입이며, 이전 결과를 활용하지 않는다.

```
/*
iterate()
*/

Stream<integer> evenStream = Stream.iterate(0, n-> n+2);


/*
generate()
*/
Stream<Double> randomStream = Stream.generate(Math::random);
Stream<Integer> oneStream  = Stream.generate(() -> 1);

/*
위 두 람다식은 기본형 스트림 타입의 참조변수로 다룰 수 없기 때문에 아래와 같이 변환을 해야 한다.
IntStream evenStream = Stream.iterate(0, n -> (n+2)).mapToInt(Integer::valueOf);
Stream<Integer> stream = evenStream.boxed();      // IntStream -> Stream<integer>
```

▼ 빈 스트림, 두 스트림의 연결

```
Stream emptyStream = Stream.empty();
long count = emptyStream.count();        //0


String[] str1 = {"123","456"};
String[] str2 = {"789","101112"};

Stream<String> str1 = Stream.of(str1);
Stream<String> str2 = Stream.of(str2);
Stream<STring> str3 = Stream.concat(str1,str2);  //두 스트림을 하나로 연결
```

▼ 스트림의 중간연산

💡 스트림 자르기 - skip, limint
스트림 요소 걸러내기 - filter(), distinct()
정렬 - sorted()
변환 - map()
조회 - peek()

```
// 자르기
IntStream,Stream<T> skip(long n)
IntStream,Stream<T> limit (long maxSize)

//요소 걸러내기
Stream<T> filter(Predicate<? super T> predicate)
Stream<T> distinct()
```

▼ 스트림- 정렬

sorted()는 지정된 Comparator로 스트림을 정렬하는데, Comparator대신 int값을 반환하는 람다식을 사용하는 것도 가능하다.
Comparator를 지정하지 않으면 스트림 요소의 기본 정렬 기준(Comparable)으로 정렬한다.
단 스트림의 요소가 Comparable을 구현한 클래스가 아니면 예외가 발생한다.

```
//정렬
Stream<T> sorted()
Stream<T> sorted(Comparator<? super T> comparator)

//기본 정렬
strStream.sorted()
strStream.sorted(Comparator.naturalOrder())
strStream.sorted(s1,s2)-> s1.compareTo(s2));
strStream.sorted(String::CompareTo);

//역순 정렬
strStream.sorted(Comparator.reverseOrder())
strStream.sorted(Comparator.<String>naturalOrder().reversed())

//대소문자 구분 안함
strStream.sorted(String.Case_INSENSITIVE_ORDER)
strStream.sorted(String.CASE_INSENSITIVE_ORDER.reversed())

//길이순 정렬
strStream.sorted(Comparator.comparing(String::length))
//no오토박싱
strStream.sorted(Comparator.comparingInt(String::length)

//길이순 오름차순
strStream.sorted(Comparator.comparing(String::length).reversed())

/** JDK1.8부터 static메서드와 디폴트 메서드가 많이 추가되었는네 이를 이용하면정렬이 쉬워진다.

        // 무게 오름차순 정렬
        inventory.sort(Comparator.comparing(Apple::getWeight));

        // 무게 내림차순 정렬
        inventory.sort(Comparator.comparing(Apple::getWeight).reversed());

        // 무게 오름차순 정렬 (무게가 같으면 색 오름차순 정렬)
        inventory.sort(Comparator.comparing(Apple::getWeight).thenComparing(Apple::getColor));

        // 무게 오름차순 정렬 (무게가 같으면 색 내림차순 정렬)
        Comparator<Apple> reversedColorComparator = Comparator.comparing(Apple::getColor).reversed();
        inventory.sort(Comparator.comparing(Apple::getWeight).thenComparing(reversedColorComparator));
```

▼ 스트림 - 변환(map)

💡 스트림의 요소에 저장된 값 중에서 원하는 필드만 뽑아내거나 특정 형태로 변환해야 할 때 사용한다.
map또한 중간연산이므로 String을 요소로하는 스트림이다.
map()도 filter()처럼 하나의 스트림에 열 번 적용할 수 있다.

기본형 스트림 IntStream,LongStream,DoubleStream을 Stream<T>로 변환할 때에는 mapToObj()를, Stream<Integer>로 변환할 때는 boxed()를 사용하낟.

```
Stream<R> map(Function<? super T, ? extends R> mapper)

DoubleStream mapToDouble(ToDoubleFunction<? super T>mapper)
IntStream mapToInteger(ToIntFunction<? super T>mapper)
LongStream mapToLong(ToLongFunction<? super T> mapper)

//예시
IntStream studentScoreStream = studentStream.mapToInt(Student::getTotalScore);
int allTotalScore = studentScoreStream.sum();
```

▼ IntSummaryStatics

💡 sum(), average()등 통계함수를 사용할 수 있다.

```
IntSummaryStatistics stat = scoersStream.summaryStatistics();

long totalCount = stat.getCount();
long totalScore = stat.getSum();
double avgScore = stat.getAverage();
int minScore - stat.getMin();
int maxScroe = stat.getMax();
```

▼ 스트림의 변환

```
1. 스트림 -> 기본형 스트림
Stream<T> -> IntStream,DoubleStream,LongStream

==변환메서드
mapToInt,mapToDouble,mapToLong(ToIntFunction<T> mapper/ ToDoubleFunction<T> mapper/ ToLongFunction<T> mapper)

2. 기본형 -> 스트림
Int/Long/DoubleStream -> stream<Integer/Long/Double> 또는 Stream<U>

==변환메서드
boxed()
mapToObj(DoubleFunction<U> mapper)

3. 기본형 스트림 -> 기본형 스트림

Int/Long/DoubleStrema -> LongStream,DoubleStream

==변환메서드
asLongStream(), asDoubleStream()

4. 스트림의 스트림 -> 스트림
```

```
Stream<Stream<T>> -> Stream<T>
Stream<IntSteram/LongStream/DoubleStream> -> IntStream,LongStream,DoubleStream

==변환메서드
flatMap, flatMapToInt/Long/Double(Function mapper)


5. 스트림 -> 컬렉션

Stream<T>,intStream,LongStream,DoubleStream -> Collection<T>, List<T>, Set<T>

==변환 메서드
Collect(Collectors.toCollection(Supplier factory));
collect(Collectors.toList());
collect(Collectors.toSet());

6. 스트림 -> Map

Stram<T>, Int/Long/DoubleStream - > MAP<K,V>

==변환 메서드
collect(Collectors.toMap(Function key, Function value))
collect(Collectors.toMap(Function k, Function v, BinaryOperator))
collect(Collectors.toMap(Function k, Function v, BianryOperator merge, Supplier mapSupplier))
```