세종대학교

Data Engineering

데이터엔지니어링

[Stream을 통한 데이터 엔지니어링]

Stream에서 활용하는 Functional Interface

학습내용

1. Stream에서 활용하는 Functional Interface 소개
2. Stream에서 활용하는 Functional Interface 실습

학습목표

- Stream에서 활용하는 Functional Interface의 개념을 설명할 수 있다.
- 학습한 Functional Interface를 이용하여 Stream method 일부를 활용할 수 있다.

Stream의 Functional Interface

Functional Interface	활용 Method	Description	Stream method 예시
Consumer〈E〉	void accept(E e)	입력 e를 받아 활용하고 반환하지 않음	forEach
Predicate〈E〉	boolean test(E e)	입력 e에 대한 테스트를 수행하고 true/false를 반환	filter
Function〈T, R〉	R apply(T t)	입력 t를 받아 R타입의 값을 반환	map
IntFunction〈R〉	R apply(int value)	int 입력 value를 받아 R타입의 값을 반환	toArray
Comparator〈T〉	int compare (T o1, T o2)	o1과 o2의 크기를 비교하여 반환 (양수, 0, 음수)	sort
ToIntFunction〈T〉	int applyAsInt (T value)	입력 t를 받아 int 로 반환	mapToInt
BiConsumer〈T, U〉	void accept(T t, U u)	입력 t와 u를 받아 활용하고 반환 값은 없음	collect
Supplier〈E〉	E get()	E타입의 값을 반환	collect
BinaryOperator〈E〉	E apply (E e1, E e2)	같은 타입 E의 입력 e1, e2를 받아 E타입의 값을 반환	reduce

Consumer

Functional Interface	활용 Method	Description	Stream method 예시
Consumer<E>	void accept(E e)	입력 e를 받아 활용하고 반환하지 않음	forEach

terminal method Stream의 각 instance를 소비함

```
Stream<Integer> s = Stream.of(1, 2, 3, 4, 5);

s.forEach(new Consumer<Integer>() {
  @Override
  public void accept(Integer t) {
    System.out.println(t);
  }
});
```

Predicate

Functional Interface	활용 Method	Description	Stream method 예시
Predicate<E>	boolean test (E e)	입력 e에 대한 테스트를 수행하고 true/false를 반환	filter

intermediate method Stream의 각 instance를 test하여 true를 반환해야 유지함

```
Stream<Integer> s = Stream.of(1, 2, 3, 4, 5);

s.filter(new Predicate<Integer>() {
  @Override
  public boolean test(Integer t) {
    if (t % 2 == 0)
      return true;
    else
      return false;
  }
}).forEach(e -> System.out.println(e));
```

Function

Functional Interface	활용 Method	Description	Stream method 예시
Function 〈T, R〉	R apply(T t)	입력 t를 받아 R타입의 값을 반환	map

intermediate method Stream의 각 instance를 이용하여 다른 값으로 매핑하여 반환

```
Stream<Integer> s = Stream.of(1, 2, 3, 4, 5);

s.map(new Function<Integer, Integer>() {
  @Override
  public Integer apply(Integer t) {
    return t + 3;
  }
}).forEach(e -> System.out.println(e));
```

Comparator

Functional Interface	활용 Method	Description	Stream method 예시
Comparator 〈T〉	int compare (T o1, T o2)	o1과 o2의 크기를 비교하여 반환 (양수, 0, 음수)	sorted

intermediate method Stream의 instance들을 정렬함

```
s.sorted(new Comparator<Integer>() {
  @Override
  public int compare(Integer o1, Integer o2) {
    if (o1 < o2)
      return -1;
    else if (o1 == o2)
      return 0;
    else
      return +1;
  }
}).forEach(System.out::println);
```

ToIntFunction

Functional Interface	활용 Method	Description	Stream method 예시
ToIntFunction<T>	int applyAsInt (T value)	입력 t를 받아 int 로 반환	mapToInt

intermediate method

```
Stream<Integer> s = Stream.of(1, 2, 3, 4, 5);

int sum = s.mapToInt(new ToIntFunction<Integer>() {
  @Override
  public int applyAsInt(Integer value) {
    return value;
  }
}).sum();
System.out.println(sum);
```

**각 인스턴스가 integer로 보장되므로,
추가적인 기능성 제공**

Supplier, BiConsumer

Functional Interface	활용 Method	Description	Stream method 예시
BiConsumer 〈T, U〉	void accept (T t, U u)	입력 t와 u를 받아 활용하고 반환 값은 없음	collect
Supplier〈E〉	E get()	E타입의 값을 반환	collect

collect	• prev = last; • item = e; • next = null;

다음 3가지 방법을 정의해야 함

Supplier	정보를 담을 문자열 생성기 StringBuilder의 생성
Accumulator	단일 값을 StringBuilder에 담는 방법 정의
Combiner	StringBuilder들을 병합하는 방법 정의 (병렬처리 시 활용)

Supplier

1	2	3	4	5	6	7

StringBuilder

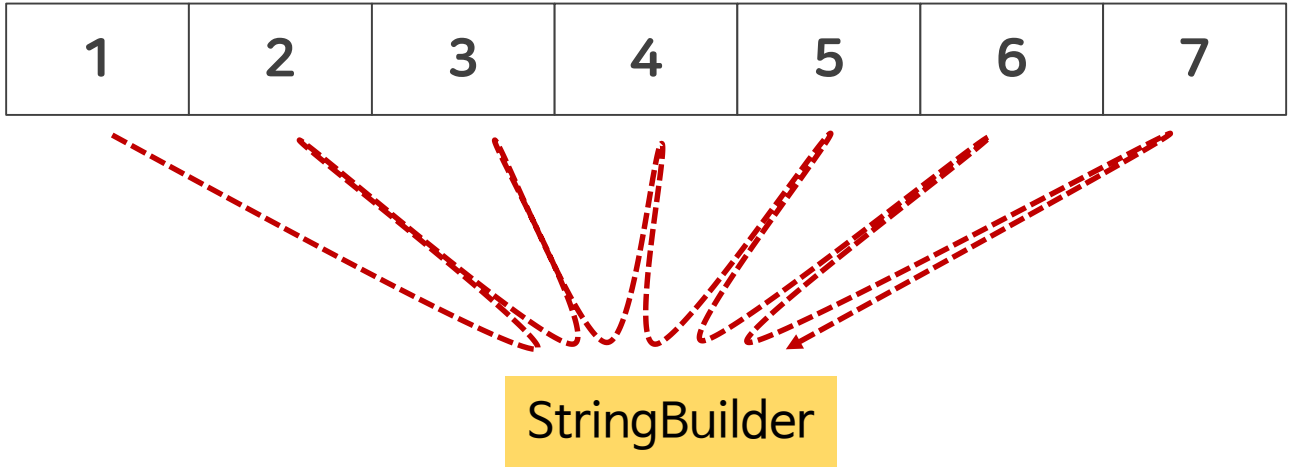
StringBuilder 생성 (Immutable한 String으로 처리 불가)	append로 문자열을 추가가능 toString으로 생성된 문자열 반환

```
() -> {
return new StringBuilder();
}
```

혹은

```
StringBuilder::new
```

Accumulator



StringBuilder에 각 instance를 추가

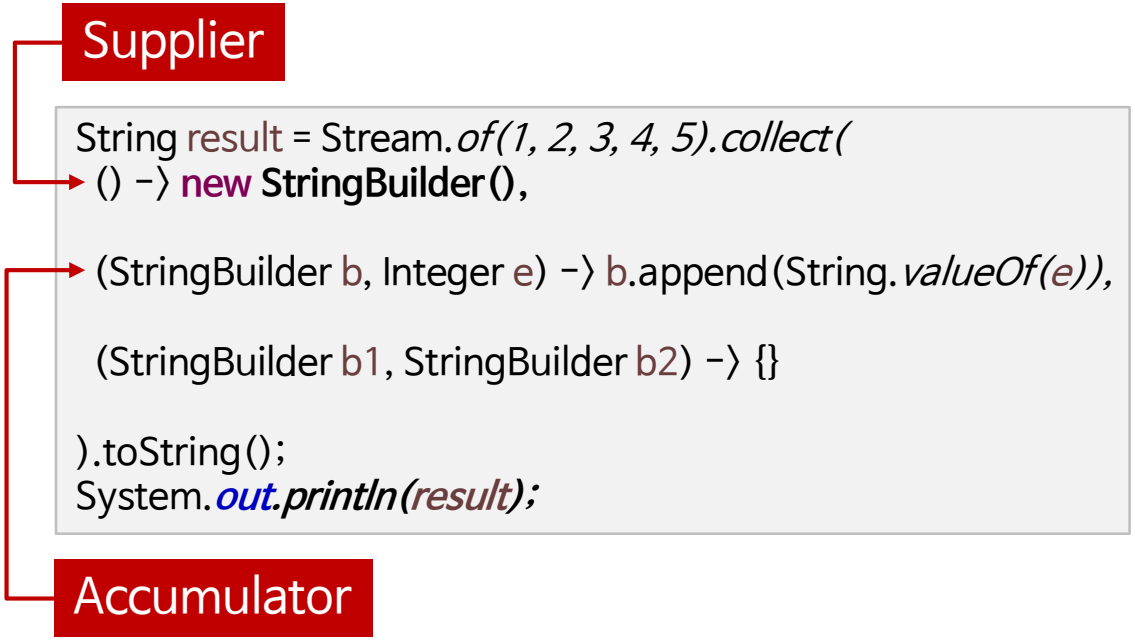
append로 문자열을 추가가능
toString으로 생성된 문자열 반환

```
new BiConsumer<StringBuilder, Integer>() {
 @Override
 public void accept(StringBuilder b, Integer e) {
  b.append(e);
 }
};1
```

혹은

```
(StringBuilder b, Integer e) ->
b.append(String.valueOf(e))
```

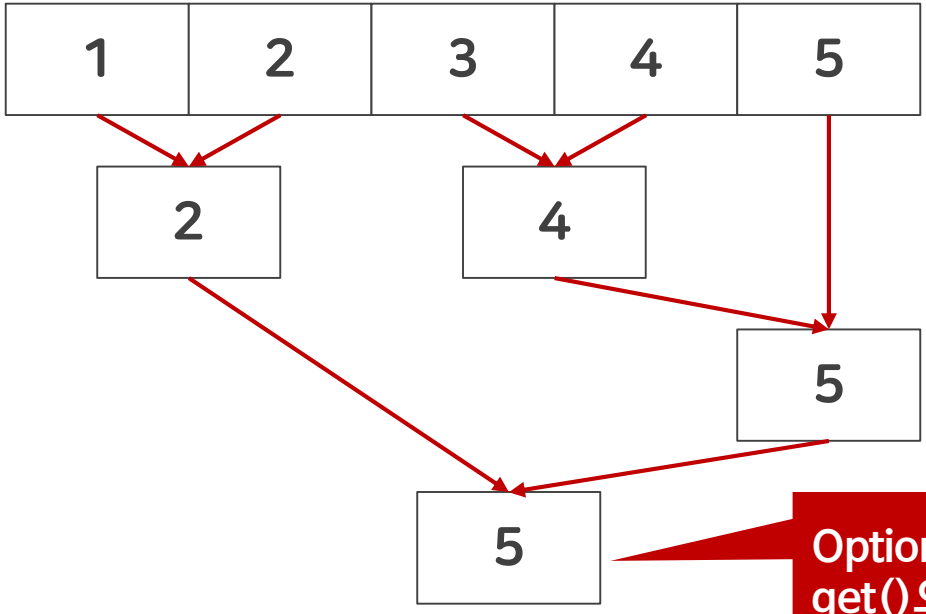
Combiner


Supplier
String result = Stream.of(1, 2, 3, 4, 5).collect(
() -> new StringBuilder(),
(StringBuilder b, Integer e) -> b.append(String.valueOf(e)),
(StringBuilder b1, StringBuilder b2) -> {}
).toString();
System.out.println(result);
Accumulator

BinaryOperator

Functional Interface	활용 Method	Description	Stream method 예시
BinaryOperator	R apply (T t, U u)	같은 타입 E의 입력 e1, e2를 받아 E타입의 값을 반환	reduce

최대값 찾기



BinaryOperator	accumulator는 둘 중 큰 값을 반환

최대값 찾기

```
Integer max = Stream.of(1, 2, 3, 4, 5).reduce(new BinaryOperator<Integer>() {
 @Override
 public Integer apply(Integer t, Integer u) {
  if (t > u) return t;
  else       return u;
 }
}).get();
System.out.println(max);
```

Remind

Stream의 Functional Interface 소개 및 실습