

스트림 활용 5장

이 장의 내용

- 필터링, 슬라이싱, 매칭
- 검색, 매칭, 리듀싱
- 특정 범위의 숫자와 같은 숫자 스트림 사용하기
- 다중 소스로부터 스트림 만들기
- 무한 스트림

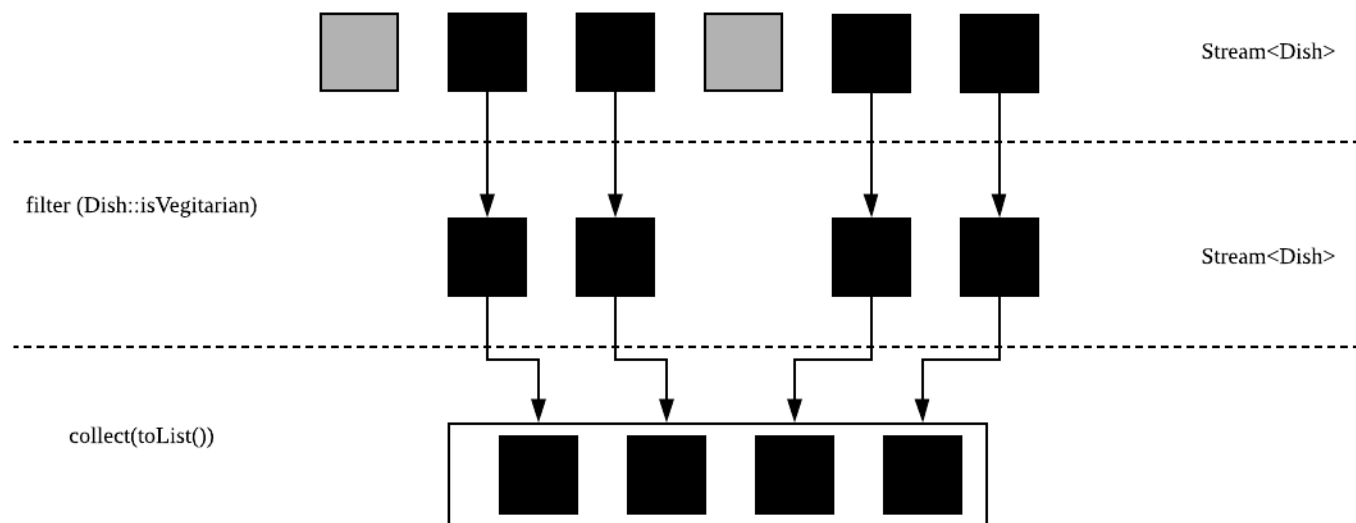
5.1 필터링

프레디케이트로 필터링

- 프레디케이트(불리언을 반환하는 함수)를 인수로 받아서 일치하는 모든 요소의 스트림을 반환

Ex

```
List<Dish> vegetarianMenu = menu.stream()  
    .filter(Dish::isVegetarian)  
    .collect(toList());
```



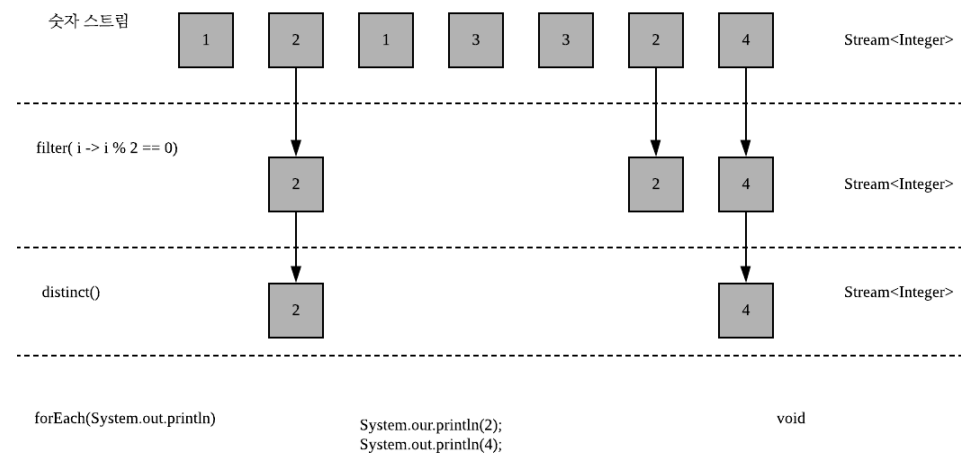
5.1 필터링

고유 요소 필터링

- Distinct 메서드를 통해 고유 요소로 이루어진 스트림을 반환

Ex

```
List<Integer> numbers = Arrays.asList(1,2,1,3,3,2,4);  
numbers.stream()  
    .filter(i -> i % 2 == 0)  
    .distinct()  
    .forEach(System.out::println);
```



5.2 스트림 슬라이싱

프레디케이트를 이용한 슬라이싱 - TAKEWHILE 활용

- 조건을 만족하는 동안만 작업을 반복한다.

```
List<Dish> specialMenu = Arrays.asList(  
    new Dish("seasonal fruit", true, 120, Dish.Type.OTHER),  
    new Dish("parawns", false, 300, Dish.Type.FISH),  
    new Dish("rice", true, 350, Dish.Type.OTHER),  
    new Dish("chicken", false, 400, Dish.Type.MEAT),  
    new Dish("french fries", true, 530, Dish.Type.OTHER)  
);
```

Ex

```
List<Dish> filteredMenu  
    = specialMenu.stream()  
        .takewhile(dish -> dish.getCalories() < 320)  
        .collect(toList());    seasonal fruit, prawns
```

5.2 스트림 슬라이싱

프레디케이트를 이용한 슬라이싱 - DROPWHILE 활용

- 조건을 만족하는 동안만 작업을 반복한다.
- 거짓이 되는 지점까지의 요소를 버린 나머지 요소를 반환한다

```
List<Dish> specialMenu = Arrays.asList(  
    new Dish("seasonal fruit", true, 120, Dish.Type.OTHER),  
    new Dish("parawns", false, 300, Dish.Type.FISH),  
    new Dish("rice", true, 350, Dish.Type.OTHER),  
    new Dish("chicken", false, 400, Dish.Type.MEAT),  
    new Dish("french fries", true, 530, Dish.Type.OTHER)  
);
```

Ex

```
List<Dish> filteredMenu  
    = specialMenu.stream()  
        .dropwhile(dish -> dish.getCalories() < 320)  
        .collect(toList());    rice, chicken, French fries
```

5.2 스트림 슬라이싱

스트림 축소 – limit(n)

- 최대 n개의 요소를 반환한다.

```
List<Dish> specialMenu = Arrays.asList(  
    new Dish("seasonal fruit", true, 120, Dish.Type.OTHER),  
    new Dish("parawns", false, 300, Dish.Type.FISH),  
    new Dish("rice", true, 350, Dish.Type.OTHER),  
    new Dish("chicken", false, 400, Dish.Type.MEAT),  
    new Dish("french fries", true, 530, Dish.Type.OTHER)  
);
```

```
List<Dish> filteredMenu  
    = specialMenu.stream()  
        .filter(dish -> dish.getCalories() > 300)  
        .limit(3);  
        .collect(toList());
```

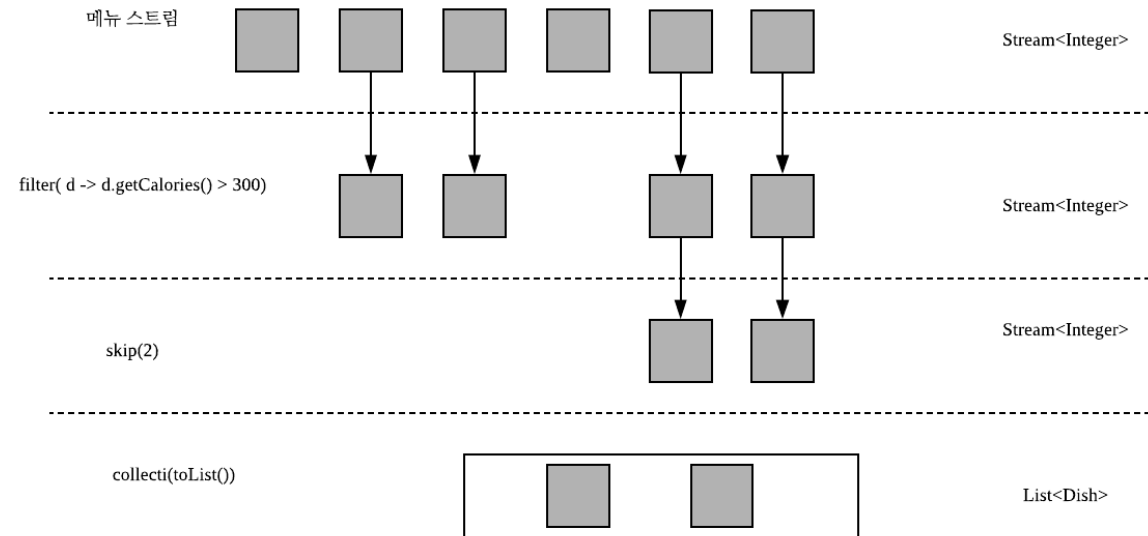
rice, chicken, French fries

5.2 스트림 슬라이싱

요소 건너뛰기 – skip(n)

- n개 이하의 요소에 대해서는 건너뛴다.

```
List<Dish> dishes  
    = specialMenu.stream()  
        .filter(d -> d.getCalories() > 300)  
        .skip(2)  
        .collect(toList());
```



5.2 스트림 슬라이싱

퀴즈 - 스트림을 이용해서 처음 등장하는 두 고기 요리를 필터링하시오.

```
public class Dish {  
    private String name;  
    private boolean vegetarian;  
    private int calorie;  
    private Type type;  
}
```

```
List<Dish> specialMenu = Arrays.asList(  
    new Dish("seasonal fruit", true, 120, Dish.Type.OTHER),  
    new Dish("parawns", false, 300, Dish.Type.MEAT),  
    new Dish("rice", true, 350, Dish.Type.MEAT),  
    new Dish("chicken", false, 400, Dish.Type.MEAT),  
    new Dish("french fries", true, 530, Dish.Type.OTHER)  
);
```

5.2 스트림 슬라이싱

정답

퀴즈

```
List<Dish>dishes =  
    specialMenu.stream()  
        .filter(d -> d.getType() == Dish.Type.MEAT)  
        .limit(2)  
        .collect(toList())
```

5.3 매핑

Map 메서드 활용

- 인수로 제공된 함수를 각 요소에 적용해 새로운 요소로 매핑 시킨다.
- 기존의 값을 고친다 라기보다는 새로운 버전을 만든다 라는 개념에 가까우므로 매핑이라는 단어 사용.

```
List<String> dishNames = specialMenu.stream()
    .map(Dish::getName)
    .collect(toList());
```

만약 요리명의 길이를 알고 싶다면??

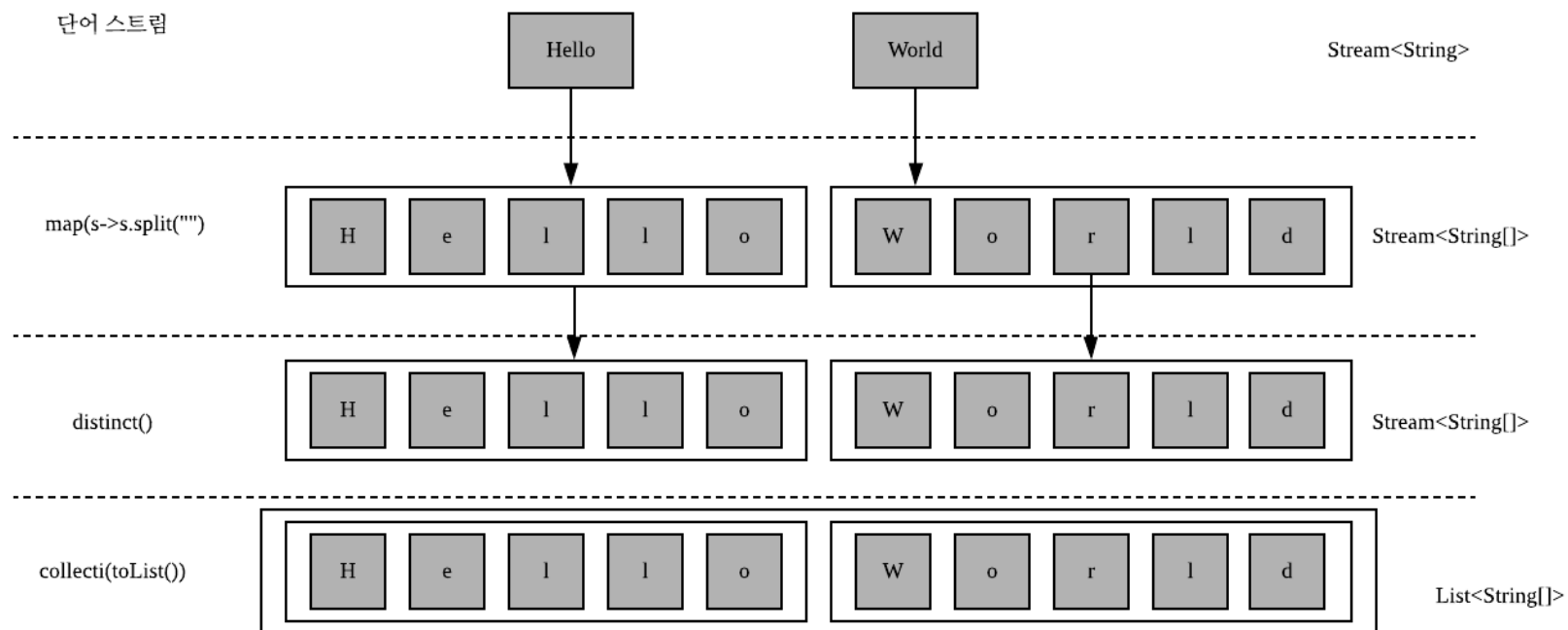
```
List<Integer> dishNameLengths = specialMenu.stream()
    .map(Dish::getName)
    .map(String::length)
    .collect(toList());
```

5.3 매핑

flatMap 메서드 활용

- 요소 -> 복수 개의 요소들로 구성된 새로운 스트림
- ["Hello", "World"] 리스트 -> ["H", "e", "l", "l", "o", "W", "r", "d"]
- 리스트에 있는 각 단어를 문자로 매핑한 후 distinct로 중복을 제거하면 되지 않을까?

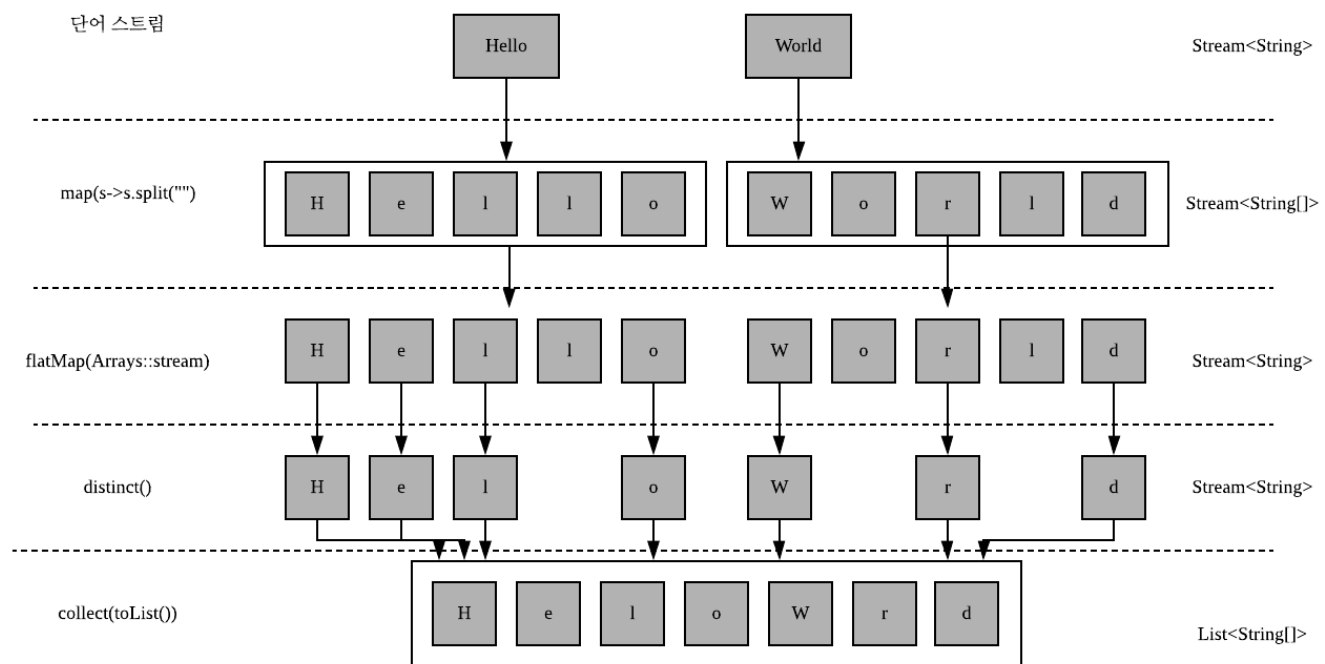
```
words.stream()
    .map(word -> word.split(""))
    .distinct()
    .collect(toList())
```



5.3 매핑

flatMap 메서드 활용

```
List<String> uniqueCharacters =  
    words.stream()  
        .map(word -> word.split(""))  
        .flatMap(Arrays::stream)  
        .distinct()  
        .collect(toList());
```



5.3 매핑

퀴즈1 – 숫자 리스트가 주어졌을 때 각 숫자의 제곱으로 이루어진 리스트를 반환하시오.

[1,2,3,4,5] 가 주어지면 [1,4,9,16,25]

정답

```
List<Integer> numbers = Arrays.asList(1,2,3,4,5);  
List<Integer> squares = numbers.stream()  
                                .map(n->n*n)  
                                .collect(toList());
```

5.3 매핑

퀴즈2 – 두 개의 숫자 리스트가 있을 때 모든 숫자 쌍의 리스트를 반환하시오.

[1,2,3] 과 [3,4] -> [(1,3) ,(1,4), (2,3) ,(2,4), (3,3), (3,4)]반환

정답

```
List<Integer> numbers1 = Arrays.asList(1,2,3);
List<Integer> numbers2 = Arrays.asList(3,4);
List<int[]> pairs = numbers1.stream()
    .flatMap(i -> numbers2.stream()
        .map(j -> new int[]{i,j}))
    .collect(toList());
```

5.3 매핑

퀴즈3 – 이전 예제에서 합이 3으로 나누어떨어지는 쌍만 반환하려면 어떻게 해야 할까?

(2,3), (3,3) 반환

정답

```
List<Integer> numbers1 = Arrays.asList(1,2,3);
List<Integer> numbers2 = Arrays.asList(3,4);
List<int[]> pairs =
    numbers1.stream()
        .flatMap(i ->
            numbers2.stream()
                .filter(j -> (i+j) % 3 == 0)
                .map(j -> new int[]{i,j}))
        .collect(toList());
```


5.4 검색과 매칭

anyMatch 메서드 활용

- 적어도 한 요소가 주어진 프레디케이트와 일치하는지 확인
- 불리언을 반환하므로 최종 연산이다.

```
if(menu.stream().anyMatch(Dish::isVegetarian)) {  
    System.out.println("The menu is (somewhat) vegetarian friendly!!");  
}
```

5.4 검색과 매칭

allMatch 메서드 활용

- 스트림의 모든 요소가 주어진 프레디케이트와 일치하는지 검사
- 불리언을 반환하므로 최종 연산이다.

allMatch

```
boolean isHealthy = menu.stream()  
    .allMatch(dish -> dish.getCalories() < 1000);
```

5.4 검색과 매칭

noneMatch 메서드 활용

- 주어진 프레디케이트와 일치하는 요소가 없는지 확인한다.

noneMatch

```
boolean isHealthy = menu.stream()
    .noneMatch(d -> d.getCalories() >= 1000);
```

5.4 검색과 매칭

쇼트서킷

- 전체 스트림을 처리하지 않아도 결과를 반환할 수 있는데 이런 상황을 쇼트 서킷이라고 부른다.
- allMatch, noneMatch, findFirst, findAny, limit 등의 연산은 원하는 요소를 찾으면 즉시 결과를 반환할 수 있는데 이를 쇼트서킷 연산이라고 한다.

5.4 검색과 매칭

findAny 메서드 활용

- 현재 스트림에서 임의의 요소를 반환한다.

findAny

```
Optional<Dish> dish =  
    menu.stream()  
        .filter(Dish::isVegetarian)  
        .findAny();
```

Optional 이란?

- 값의 존재나 부재 여부를 표현하는 컨테이너 클래스
- findAny는 아무 요소를 반환하지 않을 수 있다.
- 그러한 상황에서 null 확인 관련 버그를 피하는 방법이다.

5.4 검색과 매칭

Optional 제공 메서드

- isPresent() 값이 존재하면 true, 아니면 false 반환
- ifPresent(Consumer<T> block) 값이 있으면 block 실행
- T get() 값이 존재하면 값을 반환, 없으면 NoSuchElementException
- T orElse(T orther) 값이 있으면 값을 반환, 없으면 기본값을 반환

```
Optional<Dish> dish =  
    menu.stream()  
        .filter(Dish::isVegetarian)  
        .findAny();  
    .ifPresent(dish -> System.out.println(dish.getName()));
```

값이 있으면 출력되고 없으면 아무 일도 x

5.4 검색과 매칭

findFirst 메서드 활용

- 스트림에서 첫 번째 요소를 반환한다.

findFirst

```
List<Integer> someNumbers = Arrays.asList(1,2,3,4,5);
Optional<Integer> firstSquareDivisibleByThree =
    someNumbers.stream()
        .map(n -> n*n)
        .filter(n -> n%3 == 0)
        .findFirst();    // 9 반환
```

5.5 리듀싱

reduce 메서드 활용

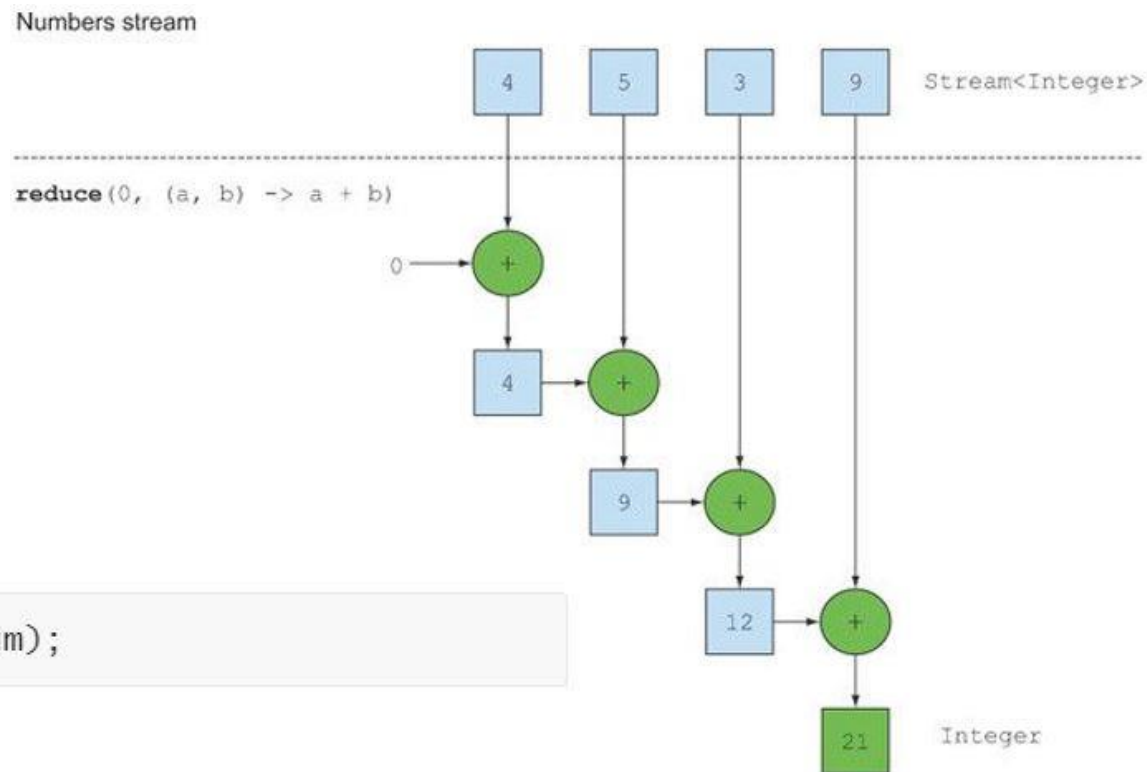
- 다양한 집계 결과물을 만들 수 있도록 해준다.

```
List<Integer> numbers = Arrays.asList(4,5,3,9);  
int sum = numbers.stream().reduce(0, (a,b) -> a+b);
```

Reduce는 두 개의 인수를 갖는다

- 초깃값
- Operator

```
int sum = numbers.stream().reduce(0, Integer::sum);
```



5.5 리듀싱

reduce 메서드 활용 – 초기값 없음

- 초기값을 받지 않도록 오버로드된 reduce도 있다.

```
Optional<Integer> sum = numbers.stream().reduce((a,b) -> (a+b));
```

- Optional<Integer> 를 반환한다.
- 스트림에 요소가 없을 때 초기값이 없으므로 합계를 반환할 수 없기 때문에 Optional 객체로 감싼 결과를 반환

5.6 실전 연습

```
public class Trader {  
  
    private final String name;  
    private final String city;  
  
    public Trader(String name, String city) {  
        this.name = name;  
        this.city = city;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getCity() {  
        return city;  
    }  
  
    @Override  
    public String toString() {  
        return "Trader:" + this.name + " in " + this.city;  
    }  
  
}
```

```
public class Transaction {  
  
    private final Trader trader;  
    private final int year;  
    private final int value;  
  
    public Transaction(Trader trader, int year, int value) {  
        this.trader = trader;  
        this.year = year;  
        this.value = value;  
    }  
  
    public Trader getTrader() {  
        return trader;  
    }  
  
    public int getYear() {  
        return year;  
    }  
  
    public int getValue() {  
        return value;  
    }  
  
    @Override  
    public String toString() {  
        return "{" + this.trader + ", " +  
            "year: " + this.year + ", " +  
            "value: " + this.value + "}";  
    }  
  
}
```

5.6 실전 연습

문제 1. 2011년에 일어난 모든 트랜잭션을 찾아서 값을 오름차순으로 정렬하시오.

문제 2. 거래자가 근무하는 모든 도시를 중복 없이 나열하시오.

문제 3. 케임브리지에서 근무하는 모든 거래자를 찾아서 이름순으로 정렬하시오.

5.7 숫자형 스트림

배경

reduce 메서드를 통해 스트림 요소의 합을 구한다

- 합계를 계산하기 전에 내부적으로 Integer를 기본형으로 언박싱한다.

```
int calories = menu.stream()
    .map(Dish::getCalories)
    .reduce(0, Integer::sum);
```

직접 sum 메서드를 호출할 수 있다면 좋지 않을까??

- 스트림에는 sum 메서드가 없다.

```
int calories = menu.stream()
    .map(Dish::getCalories)
    .sum();
```

5.7 숫자형 스트림

기본형 특화 스트림

- IntStream
- DoubleStream
- LongStream

박스 비용을 피할 수 있다.

집계를 처리할 수 있는 sum, max 등의 숫자 관련 리듀싱 연산 수행 메서드 제공

필요하다면 객체 스트림으로 복원 가능

5.7 숫자형 스트림

숫자 스트림으로 매핑

- mapToInt
- mapToDouble
- mapToLong

```
int calories = menu.stream()
                    .mapToInt(Dish::getCalories)    //IntStream 반환
                    .sum();
```

5.7 숫자형 스트림

객체 스트림으로 복원

-IntStream의 map 연산은 int를 인수로 받아서 int를 반환하는 람다를 인수로 받는다.

-정수가 아닌 Dish 같은 다른 값을 반환하고 싶다면 변환 가능!

```
IntStream intStream maxCalories = menu.stream()  
                                .mapToInt(Dish::getCalories); //스트림 -> 숫자 스트림  
Stream<Integer> stream = intStream.boxed() //숫자 스트림 -> 스트림
```

5.7 숫자형 스트림

기본값: OptionalInt

스트림에 요소가 없어서 0, 실제 최대값이 0 어떻게 구별??

```
OptionalInt maxCalories = specialMenu.stream()
    .mapToInt(Dish::getCalorie)
    .max();

int max = maxCalories.orElse(1);
```


5.7 숫자형 스트림

숫자 범위

IntStream과 LongStream에서는 range와 rangeClosed라는 두 가지 정적 메서드를 제공. rangeClosed는 시작값과 종료값이 결과에 포함되며 range는 포함되지 않는다.

```
IntStream evenNumbers = IntStream.rangeClosed(1, 100)
                                   .filter(n -> n % 2 == 0);
System.out.println(evenNumbers.count());
```

1부터 100 까지중에서 짝수를 구하는 것이므로 50 개의 결과 반환

5.8 스트림 만들기

값으로 스트림 만들기

Stream.of()

```
Stream<String> stream = Stream.of("Modern ", "Java ", "In ", "Action ");  
  
stream.map(String::toUpperCase)  
      .forEach(System.out::println);
```

Stream.empty()

```
Stream<String> stream1 = stream1.empty();
```

5.8 스트림 만들기

Null이 될 수 있는 객체로 스트림 만들기

System.getProperty()

```
String value = System.getProperty("java.version") // 12.0.2 반환
```

Stream.ofNullable()

```
String homeValue = System.getProperty("user.dir");  
Stream<String> homeValueStream  
    = homeValue == null ? Stream.empty() : Stream.of(homeValue);  
  
Stream<String> homeValueStream1  
    = Stream.ofNullable(System.getProperty("home"));
```

5.8 스트림 만들기

배열로 스트림 만들기

`Arrays.stream()` 배열을 스트림으로 만들어준다.

```
int[] numbers = {2,3,5,7,11,13};  
int sum = Arrays.stream(numbers).sum();
```

5.8 스트림 만들기

파일로 스트림 만들기

Files.lines()

-주어진 파일의 행 스트림을 문자열로 반환

```
long uniqueWords = 0;
try (Stream<String> lines =
    Files.lines(Paths.get("data.txt"), Charset.defaultCharset())) {
    uniqueWords = lines.flatMap(line -> Arrays.stream(line.split(" ")))
        .distinct()
        .count();
    System.out.println(uniqueWords);
} catch (IOException e) {
    e.printStackTrace();
}
```

5.8 스트림 만들기

함수로 무한 스트림 만들기

- 스트림 API는 함수에서 스트림을 만들 수 있는 두 정적 메서드 `Stream.iterate`와 `Stream.generate` 제공

`iterate()`

- `iterate`는 초기값과 람다를 인수로 받아서 새로운 값을 끊임없이 생산한다.

```
Stream.iterate(0, n->n+2)
    .limit(10)
    .forEach(System.out::println);
```

`generate()`

- 각 값을 연속적으로 계산하지 않는다.

```
Stream.generate(Math::random)
    .limit(5)
    .forEach(System.out::println);
```