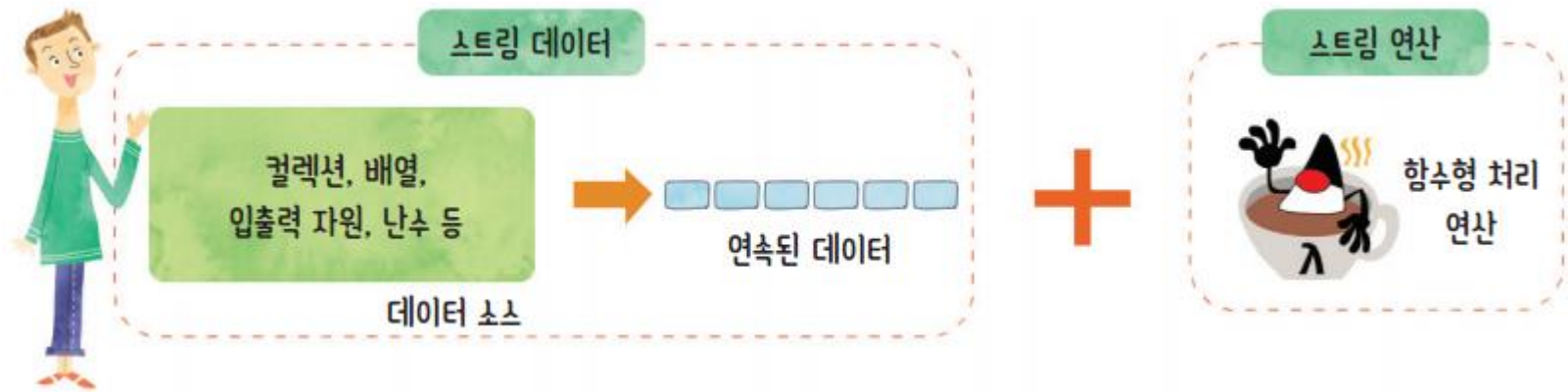


스트림

스트림 소개

■ 스트림 의미

- JDK 8부터 새롭게 추가된 기능으로 데이터 집합체를 반복적으로 처리
- 스트림을 이용하면 다수의 스레드 코드를 구현하지 않아도 데이터를 병렬로 처리
- 스트림은 스트림 데이터와 스트림 연산의 개념을 모두 포함

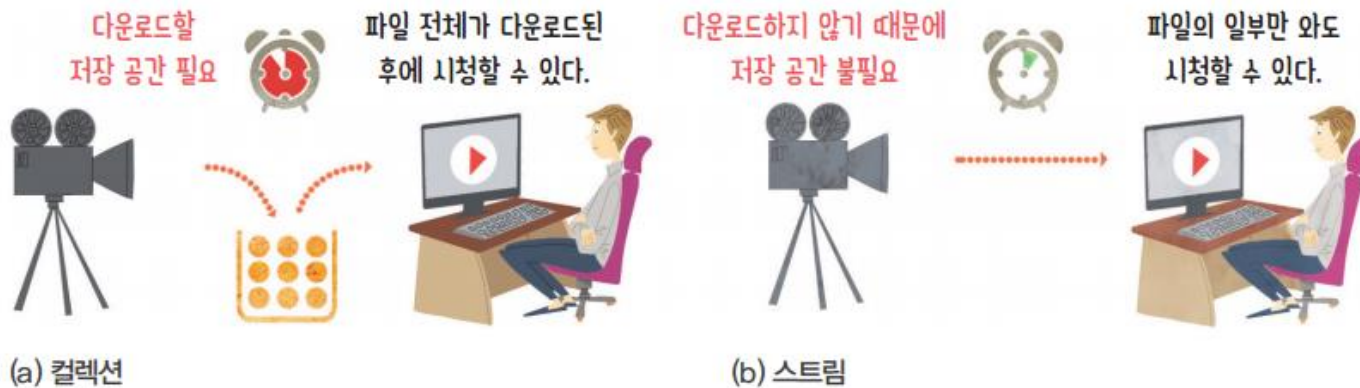


스트림 소개

■ 컬렉션과 스트림

구분	컬렉션	스트림
처리 방식	다운로드	스트리밍
저장 공간	필요	불필요
반복 방식	외부 반복	내부 반복
코드 구현	명령형	선언형
원본 데이터	변경	변경하지 않고 소비
연산 병렬화	어려움	쉬움

● 데이터 처리 방식



스트림 소개

■ 컬렉션과 스트림

- 컬렉션이 데이터의 공간적 집합체라면, 스트림은 데이터의 시간적 집합체이다.
- 컬렉션은 데이터 원소의 효율적인 관리와 접근에 맞게 설계되어 있지만, 스트림은 데이터 원소에서 수행할 함수형 연산에 맞게 설계되어 있다.
- 따라서 스트림은 원소에 직접 접근하거나 조작하는 수단을 제공하지 않는다.
- 스트림을 사용하면 코드가 간단해지고 오류 발생 확률이 줄어든다.
- 예제 : [sec01/StreamDemo](#)

```
[19, 20, 23, 24, 27, 28]
19 20 23 24 27 28
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class StreamDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        Random r = new Random();

        for (int i = 0; i < 10; i++)
            list.add(r.nextInt(30));

        List<Integer> gt10 = new ArrayList<>();
        for (int i : list)
            if (i > 10)
                gt10.add(i);

        Collections.sort(gt10);
        System.out.println(gt10);

        list.stream().filter(i -> i > 10).sorted()
            .forEach(x -> System.out.print(x + " "));
    }
}
```

스트림 소개

- 공통으로 사용될 클래스 : [sec01/Nation](#), [sec01/Util](#)

```
import java.util.List;

public class Nation {
    private final String name; private final Type type;
    private final double population; private final int gdpRank;

    public Nation(String name, Type type, double population, int gdpRank) {
        this.name = name; this.type = type;
        this.population = population; this.gdpRank = gdpRank;
    }

    public String getName() { return name; }
    public Type getType() { return type; }
    public boolean isIsland() { return getType() != Type.LAND; }
    public double getPopulation() { return population; }
    public int getGdpRank() { return gdpRank; }
    public enum Type { LAND, ISLAND }
    public String toString() { return name; }

    public static final List<Nation> nations = List.of(
        new Nation("ROK", Nation.Type.LAND, 51.4, 11),
        new Nation("New Zealand", Type.ISLAND, 4.5, 49),
        new Nation("USA", Type.LAND, 318.9, 1),
        new Nation("China", Type.LAND, 1355.7, 2),
        new Nation("Philiphine", Type.ISLAND, 107.7, 36),
        new Nation("United Kingdom", Type.ISLAND, 63.7, 5),
        new Nation("Sri Lanka", Type.ISLAND, 21.9, 63),
        new Nation("Morocco", Type.LAND, 33.0, 60)
    );
}
```

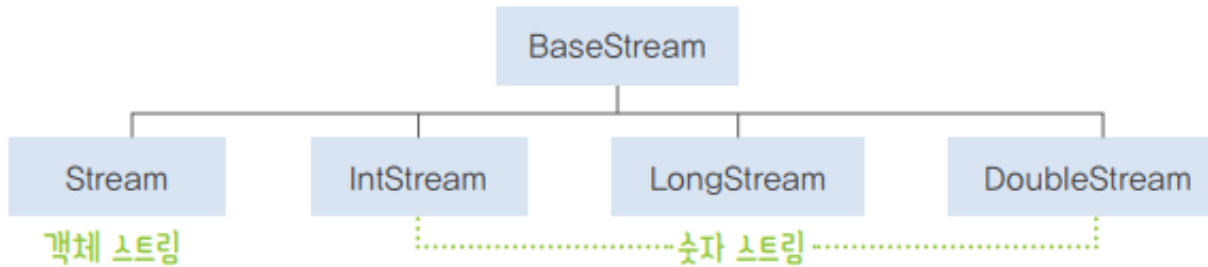
스트림 소개

- 공통으로 사용될 클래스 : [sec01/Nation](#), [sec01/Util](#)

```
public class Util {  
    public static <T> void print(T t) {  
        System.out.print(t + " ");  
    }  
  
    public static <T> void printWithParenthesis(T t) {  
        System.out.print("(" + t + ") ");  
    }  
}
```

스트림 종류와 생성

■ 스트림 종류



■ 숫자 스트림과 객체 스트림의 차이

- 숫자 스트림은 평균, 합계를 반환하는 `average()`, `sum()`이라는 메서드가 있다.
- 객체 스트림이 제공하는 최종 연산이 `Optional` 타입을 반환한다면 숫자 스트림은 `OptionalInt`, `OptionalLong`, `OptionalDouble` 타입을 반환한다.
- 숫자 스트림은 데이터 스트림의 기본 통계 요약 내용을 나타내는 `summaryStatistics()` 메서드를 제공한다.
- `IntStream`과 `Stream<Integer>` : 대체가 될 수는 있지만 성능이 떨어지고 `Average`나 `sum` 사용 불가.

스트림 종류와 생성

■ 스트림 생성

- 컬렉션으로부터 스트림 생성

```
default Stream<E> stream()  
default Stream<E> parallelStream()
```

list.stream()

- 배열로부터 스트림 생성

```
static IntStream stream(int[] array)  
static IntStream of(int... values)
```

Arrays 클래스가 제공

```
static <T> Stream of(T... values)
```

Stream 인터페이스가 제공

```
static IntStream of(int... values)
```

IntStream 인터페이스가 제공

- 예제 : [sec02/Array2StreamDemo](#)

스트림 종류와 생성

■ 스트림 생성

- 예제 : [sec02/Array2StreamDemo](#)

```
import java.util.Arrays;
import java.util.stream.DoubleStream;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class Array2StreamDemo {
    public static void main(String[] args) {
        int[] ia = {2, 3, 5, 7, 11, 13};
        IntStream is = Arrays.stream(ia);

        String[] strings = {"The", "pen", "is", "mightier", "than", "the", "sword"};
        Stream<String> ss =
            Stream.of(strings);

        double[] da = {1.2, 3.14, 5.8, 0.2};
        DoubleStream ds =
            DoubleStream.of(da);
    }
}
```

스트림 종류와 생성

■ 스트림 생성

- 기타 데이터로부터 스트림 생성
 - Random의 ints(), longs(), doubles()
 - 숫자 스트림과 객체 스트림의 iterate()와 generate()
 - IntStream과 LongStream의 range() 혹은 rangeClosed()
 - 입출력 파일이나 폴더로부터도 스트림을 생성
 - ...
- 예제 : [sec02/Etc2StreamDemo](#)

```
import java.util.Random;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class Etc2StreamDemo {
    public static void main(String[] args) {
        IntStream is1 = IntStream.iterate(1, x -> x + 2);

        IntStream is2 = new Random().ints(0, 10);

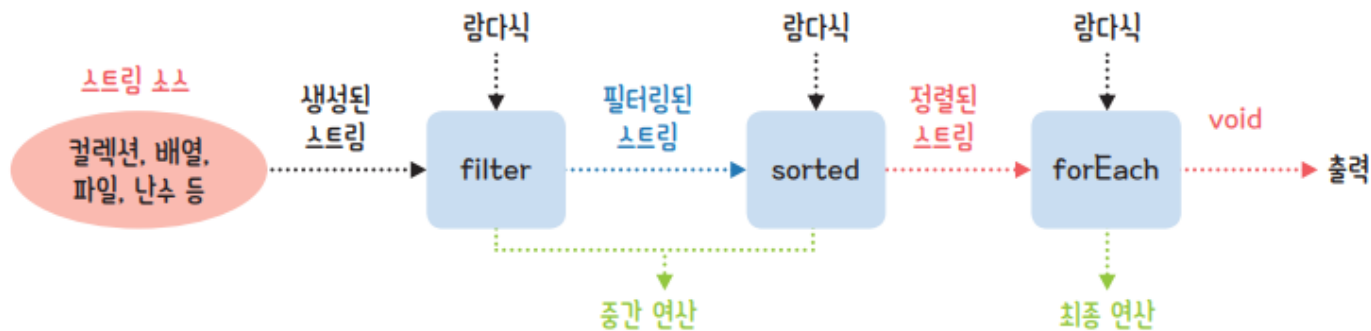
        Stream<Double> ds = Stream.generate(Math::random);

        IntStream is3 = IntStream.range(1, 5);
    }
}
```

스트림 연산과 옵션 타입

■ 스트림 파이프라인

- 스트림 연산의 결과가 Stream 타입이면 연속적으로 호출할 수 있다. 스트림 연산의 연속 호출은 여러 개의 스트림이 연결되어 스트림 파이프라인을 형성



■ 느긋한 연산과 조급한 연산

- 느긋한 연산은 조급한 연산이 데이터 소스에게 원소를 요구할 때까지 아무 연산도 수행하지 않고 기다리는 특징
- 스트림의 최종 연산은 조급한 연산이지만 중간 연산은 느긋한 연산이다.
- 최종 연산이 호출되기 전까지 중간 연산은 아무런 작업을 수행하지 않는다.
- 스트림의 중간 연산이 느긋한 연산이기 때문에 다운로드 방식처럼 저장 공간이 따로 필요 없다. 따라서 스트림 연산은 빅데이터뿐만 아니라 무한 스트림에도 대응할 수 있다.

스트림 연산과 옵션 타입

■ 느긋한 연산과 조급한 연산

- 예제 : [sec03/Laziness1Demo](#), [sec03/Laziness2Demo](#)

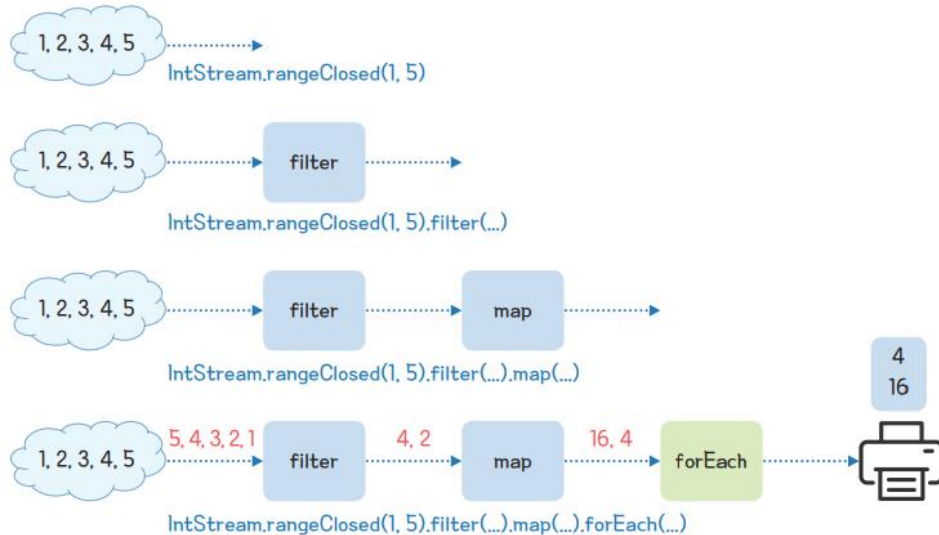
```
import java.util.stream.IntStream;

public class Laziness1Demo {
    public static void main(String[] args) {
        IntStream is = IntStream.rangeClosed(1, 5);

        is.filter(x -> {
            System.out.println("filter : " + x);
            return x % 2 == 0;
        }).map(x -> {
            System.out.println("map : " + x);
            return x * x;
        });

        /*
        }).forEach(x ->
            System.out.println("forEach : " + x)
        );
        */
    }
}
```

filter : 1
filter : 2
map : 2
forEach : 4
filter : 3
filter : 4
map : 4
forEach : 16
filter : 5



스트림 연산과 옵션 타입

■ 병렬 처리

- 멀티 코어 CPU를 제대로 활용하여 데이터 소스를 병렬로 처리할 수 있도록 병렬 스트림도 지원
- 병렬 스트림은 컬렉션 혹은 순차 스트림으로부터 각각 `parallelStream()` 혹은 `parallel()` 메서드를 호출하여 병렬 스트림 획득
- 스트림을 부분 스트림으로 분할하기 어렵거나 데이터 소스의 크기가 작거나 혹은 싱글 코어 CPU 라면 오히려 성능이 나빠질 수 있다.
- 예제 : [sec03/ParallelDemo](#)

순차 처리 : 31 병렬 처리 : 16

```
import java.util.stream.IntStream;

public class ParallelDemo {
    public static void main(String[] args) {
        long start, end, total;

        IntStream sequential = IntStream.rangeClosed(1, 100000000);
        start = System.currentTimeMillis();
        total = sequential.sum();
        end = System.currentTimeMillis();
        System.out.println("순차 처리 : " + (end - start));

        IntStream parallel = IntStream.rangeClosed(1, 100000000).parallel();
        start = System.currentTimeMillis();
        total = parallel.sum();
        end = System.currentTimeMillis();
        System.out.println("병렬 처리 : " + (end - start));
    }
}
```

스트림 연산과 옵션 타입

■ 옵션 타입

- String과 Date는 상속 관계가 아닌데도 null ?

```
String s = null;  
Date d = null;
```

- java.util 패키지 소속
- Optional은 null을 사용하지 않고 부재 값을 포함한 데이터를 저장하는 클래스로써 값의 존재 여부에 따라 다양하게 처리할 수 있는 기능을 제공
- 종류 : Optional, OptionalInt, OptionalLong, OptionalDouble

스트림 연산과 옵션 타입

■ 옵션 타입

- Optional 및 OptionalInt 클래스가 제공하는 주요 연산(OptionalInt는 파란색)

메서드	의미
static Optional empty()	빈 Optional 객체를 반환한다.
T get(), int getAsInt()	값을 반환하며, 없다면 예외를 던진다.
boolean isPresent()	값의 존재 여부를 반환한다.
void ifPresent(Consumer)	값이 있을 때 Consumer에서 처리한다.
static Optional of(T) static OptionalInt of(int)	주어진 값을 가진 Optional 타입으로 변환한다.
static Optional ofNullable(T) static Optional ofNullable(int)	주어진 값을 가진 Optional을 반환한다. 만약 null이라면 빈 Optional을 반환한다.
T orElse(T), int orElse(int)	값이 없을 때 디폴트 값을 지정한다.

스트림 연산과 옵션 타입

■ 옵션 타입

- 예제 : [sec03/Optional1Demo](#)

```
import java.util.Optional;
import java.util.OptionalDouble;
import java.util.OptionalInt;

public class Optional1Demo {

    public static OptionalDouble divide(double x, double y) {
        return y == 0 ? OptionalDouble.empty() : OptionalDouble.of(x / y);
    }

    public static void main(String[] args) {
        OptionalInt i = OptionalInt.of(100);
        OptionalDouble d = OptionalDouble.of(3.14);
        Optional<String> s = Optional.of("apple");

        System.out.println(i.getAsInt());
        System.out.println(d.getAsDouble());
        System.out.println(s.get());

        System.out.println(i);
        System.out.println(d);
        System.out.println(s);

        System.out.println(divide(1.0, 2.0));
        System.out.println(divide(1.0, 0.0));
    }
}
```

```
100
3.14
apple
OptionalInt[100]
OptionalDouble[3.14]
Optional[apple]
OptionalDouble[0.5]
OptionalDouble.empty
```


스트림 연산과 옵션 타입

- 예제 : [sec03/Optional2Demo](#)

```
import java.util.Optional;

public class Optional2Demo {
    public static void main(String[] args) {
        String s1 = "안녕"; // or String s1 = null;
        Optional<String> o = Optional.ofNullable(s1);

        if(s1 != null)
            Util.print(s1);
        else
            Util.print("없음");

        if(o.isPresent())
            Util.print(o.get());
        else
            Util.print("없음");

        String s2 = o.orElse("없음");
        Util.print(s2);

        o.ifPresentOrElse(Util::print, () -> System.out.println("없음"));
    }
}
```

- s1 = "안녕!"인 경우

안녕! 안녕! 안녕! 안녕!

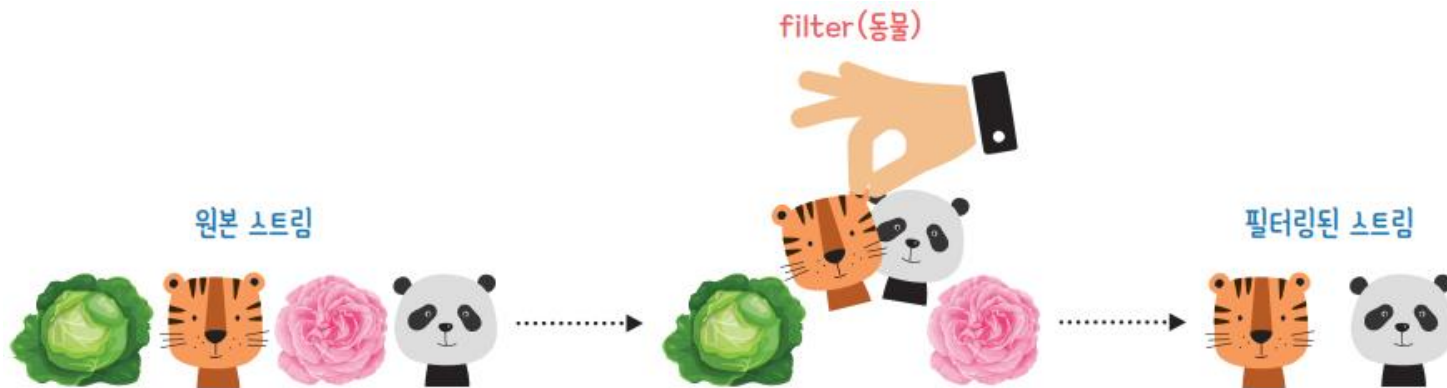
- s1 = null인 경우

없음 없음 없음 없음

스트림 활용

■ 필터링

- 입력된 스트림 원소 중에서 일부 원소를 걸러내는 중간 연산
- 필터링 연산은 filter(), distinct(), limit(), skip()



스트림 활용

■ 필터링

- 예제 : [sec04/FilterDemo](#)

문자열 스트림 : c2 c3

정수 스트림 : 2 4

인구가 1억(100백만) 이상의 2개 나라 : (USA) (China)

```
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class FilterDemo {
    public static void main(String[] args) {
        Stream<String> s1 = Stream.of("a1", "b1", "b2", "c1", "c2", "c3");
        Stream<String> s2 = s1.filter(s -> s.startsWith("c"));
        Stream<String> s3 = s2.skip(1); // 첫번째 원소 스킵
        System.out.print("문자열 스트림 : ");
        s3.forEach(Util::print);

        IntStream i1 = IntStream.of(1, 2, 1, 3, 3, 2, 4);
        IntStream i2 = i1.filter(i -> i % 2 == 0);
        IntStream i3 = i2.distinct(); // 중복값 제거
        System.out.print("\n정수 스트림 : ");
        i3.forEach(Util::print);

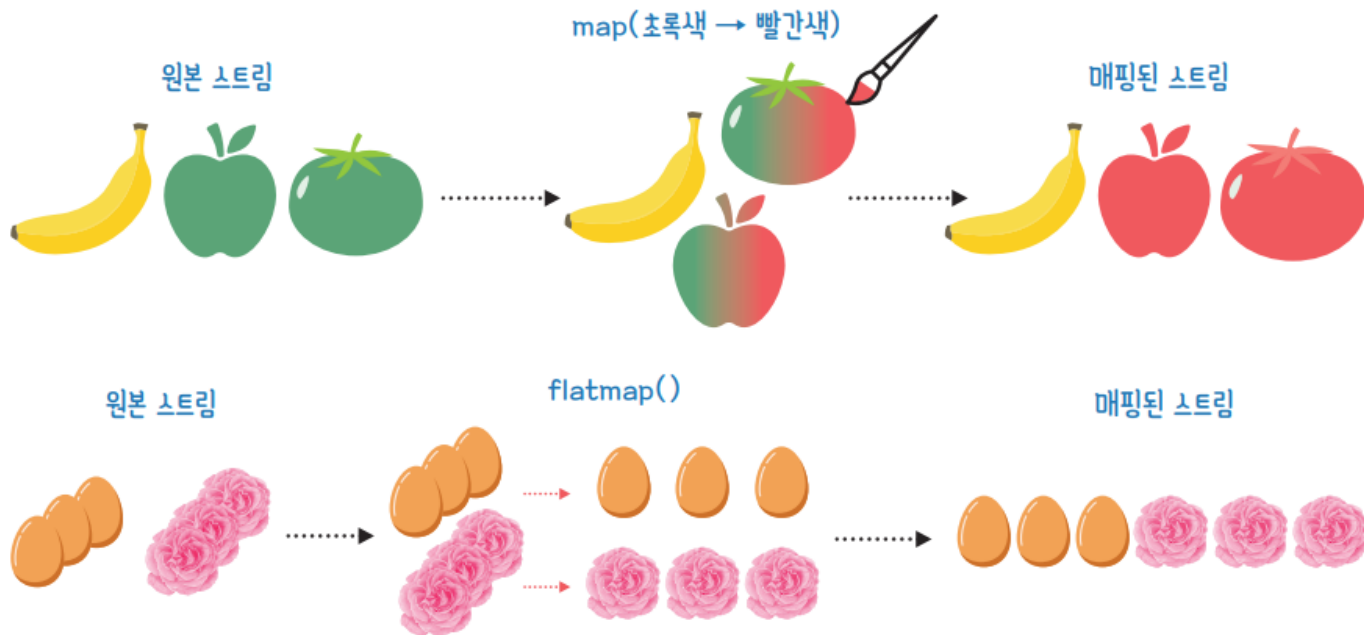
        System.out.print("\n인구가 1억(100백만) 이상의 2개 나라 : ");
        Stream<Nation> n1 = Nation.nations.stream();
        Stream<Nation> n2 = n1.filter(p -> p.getPopulation() > 100.0);
        Stream<Nation> n3 = n2.limit(2); // 2개 나라만
        n3.forEach(Util::printWithParenthesis);
    }
}
```

```
Stream.of("a1", "b1", "b2", "c1", "c2", "c3")
    .filter(s -> s.startsWith("c"))
    .skip(1).forEach(Util::print);
```

스트림 활용

■ 매핑

- 매개 값으로 제공된 람다식을 이용하여 입력 스트림의 객체 원소를 다른 타입 혹은 다른 객체 원소로 매핑
- Map(), flatMap(), mapToObj(), mapToInt(), asLongStram(), boxed()



스트림 활용

■ 매핑

- 예제 : [sec04/Map1Demo](#)

A1	B1	B2	C1	C2
2	4	2	6	6
4	8			
b1	b2	b3		

```
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class Map1Demo {
    public static void main(String[] args) {
        Stream<String> s1 = Stream.of("a1", "b1", "b2", "c1", "c2");

        Stream<String> s2 = s1.map(String::toUpperCase);
        s2.forEach(Util::print);
        System.out.println();

        Stream<Integer> i1 = Stream.of(1, 2, 1, 3, 3, 2, 4);

        Stream<Integer> i2 = i1.map(i -> i * 2);
        i2.forEach(Util::print);
        System.out.println();

        Stream<String> s3 = Stream.of("a1", "a2", "a3");

        Stream<String> s4 = s3.map(s -> s.substring(1));

        IntStream i3 = s4.mapToInt(Integer::parseInt);

        Stream<String> s5 = i3.mapToObj(i -> "b" + i);

        s5.forEach(Util::print);
    }
}
```

스트림 활용

■ 매핑

- 예제 : [sec04/Map2Demo](#),

(ROK) (New Zealand) (USA) (China)
11 49 1 2 36 5 63 60

```
import java.util.stream.Stream;

public class Map2Demo {
    public static void main(String[] args) {
        Stream<Nation> n1 = Nation.nations.stream();
        Stream<String> s1 = n1.map(Nation::getName);
        s1.limit(4).forEach(Util::printWithParenthesis);
        System.out.println();

        Stream<Nation> n2 = Nation.nations.stream();
        IntStream is = n2.mapToInt(Nation::getGdpRank);
        is.forEach(Util::print);
    }
}
```

스트림 활용

■ 매핑

● 예제 : [sec04/Map3Demo](#)

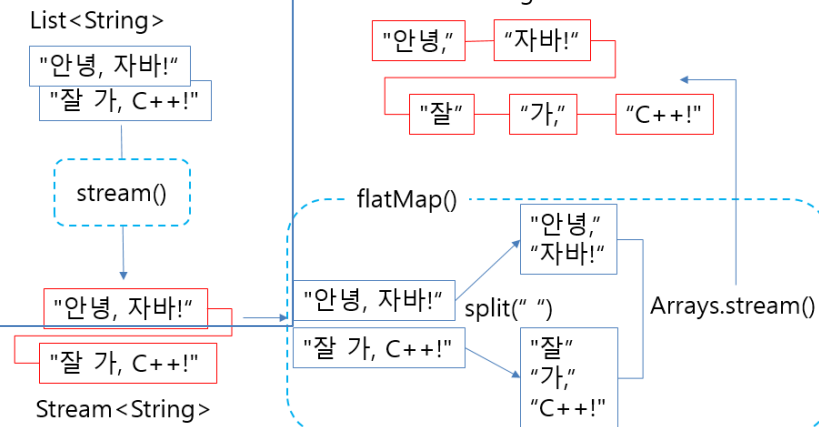
(안녕,) (자바!) (잘) (가,) (C++!)
(안녕, 자바!) (잘 가, C++!) (안녕! 람다) (환영! 스트림)

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class Map3Demo {
    public static void main(String[] args) {
        List<String> list1 = List.of("안녕, 자바!", "잘 가, C++!");
        Stream<String> s1 = list1.stream();
        Stream<String> s2 = s1.flatMap(s -> Arrays.stream(s.split(" ")));
        s2.forEach(Util::printWithParenthesis);
        System.out.println();

        List<String> list2 = List.of("좋은 아침");
        List<String> list3 = List.of("안녕! 람다", "환영! 스트림");

        Stream<List<String>> s3 = Stream.of(list1, list2, list3);
        Stream<String> s4 = s3.flatMap(list -> {
            if (list.size() > 1)
                return list.stream();
            else
                return Stream.empty();
        });
        s4.forEach(Util::printWithParenthesis);
    }
}
```



스트림 활용

■ 정렬

- 입력된 스트림 원소 전체를 정렬하는 중간 연산으로 `distinct()` 연산과 마찬가지로 버퍼가 필요

```
Stream<T> sorted()  
Stream<T> sorted(Comparator<T>)  
IntStream sorted()  
LongStream sorted()  
DoubleStream sorted()
```

모두 중간 연산이기 때문에 스트림을 반환한다.

스트림 활용

■ 정렬

- 예제 : [sec04/SortedDemo](#)

```
import java.util.Comparator;
import java.util.stream.Stream;

public class SortedDemo {
    public static void main(String[] args) {
        Stream<String> s1 = Stream.of("d2", "a2", "b1", "b3", "c");
        Stream<String> s2 = s1.sorted();
        s2.forEach(Util::print);

        System.out.print("\n국가 이름 순서 : ");
        Stream<Nation> n1 = Nation.nations.stream();
        Stream<Nation> n2 = n1.sorted(Comparator.comparing(Nation::getName));
        Stream<String> s3 = n2.map(x -> x.getName());
        s3.forEach(Util::printWithParenthesis);

        System.out.print("\n국가 GDP 순서 : ");
        Stream<Nation> n3 = Nation.nations.stream();
        Stream<Nation> n4 = n3.sorted(Comparator.comparing(Nation::getGdpRank));
        Stream<String> s4 = n4.map(Nation::getName);
        s4.forEach(Util::printWithParenthesis);
    }
}
```

a2 b1 b3 c d2

국가 이름 순서 : (China) (Morocco) (New Zealand) (Philiphine) (ROK) (Sri Lanka) (USA) (United Kingdom)

국가 GDP 순서 : (USA) (China) (United Kingdom) (ROK) (Philiphine) (New Zealand) (Morocco) (Sri Lanka)

스트림 활용

■ 매칭과 검색

- 특정 속성과 일치되는 스트림 원소의 존재 여부를 조사하거나 검색하는 데 사용되는 스트림의 최종 연산

```
boolean allMatch(Predicate<? super T> predicate)
boolean anyMatch(Predicate<? super T> predicate)
boolean noneMatch(Predicate<? super T> predicate)
Optional<T> findAny()
Optional<T> findFirst()
```

스트림 활용

■ 매칭과 검색

- 예제 : [sec04/MatchDemo](#)

```
import java.util.Optional;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class MatchDemo {
    public static void main(String[] args) {
        boolean b1 = Stream.of("a1", "b2", "c3").anyMatch(s -> s.startsWith("c"));
        System.out.println(b1);

        boolean b2 = IntStream.of(10, 20, 30).allMatch(p -> p % 3 == 0);
        System.out.println(b2);

        boolean b3 = IntStream.of(1, 2, 3).noneMatch(p -> p == 3);
        System.out.println(b3);

        if (Nation.nations.stream().allMatch(d -> d.getPopulation() > 100.0))
            System.out.println("모든 국가의 인구가 1억이 넘는다.");
        else
            System.out.println("모든 국가의 인구가 1억이 넘지 않는다.");

        Optional<Nation> nation = Nation.nations.stream().findFirst();
        nation.ifPresentOrElse(Util::print, () -> System.out.print("없음."));
        System.out.println();

        nation = Nation.nations.stream().filter(Nation::isIsland).findAny();
        nation.ifPresent(Util::print);
    }
}
```

```
true
false
false
모든 국가의 인구가 1억이 넘지 않는다.
ROK
New Zealand
```

스트림 활용

■ 루핑과 단순 집계

- 루핑은 전체 원소를 반복하는 연산으로 `forEach()`와 `peek()`가 있다.
- 그러나 `forEach()`는 최종 연산이며, `peek()`는 중간 연산이다.
- 단순 집계는 스트림을 사용하여 스트림 원소의 개수, 합계, 평균값, 최댓값, 최솟값 등과 같은 하나의 값을 도출하는 최종 연산
- 단순 집계 연산으로 `count()`, `sum()`, `average()`, `max()`, `min()` 등

스트림 활용

■ 루핑과 단순 집계

- 예제 : [sec04/LoopAggregateDemo](#)

```
package sec04;

import java.util.Comparator;
import java.util.Optional;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class LoopAggregateDemo {
    public static void main(String[] args) {
        Stream<Nation> sn =
Nation.nations.stream().peek(Util::printWithParenthesis);
        System.out.println("어디 나타날까?");
        Optional<Nation> on =
sn.max(Comparator.comparing(Nation::getPopulation));
        System.out.println();
        System.out.println(on.get());

        System.out.println(IntStream.of(5, 1, 2, 3).min().getAsInt());

        sn = Nation.nations.stream();
        System.out.println(sn.count());
    }
}
```

어디 나타날까?

(ROK) (New Zealand) (USA) (China) (Philiphine) (United Kingdom) (Sri Lanka) (Morocco)

China

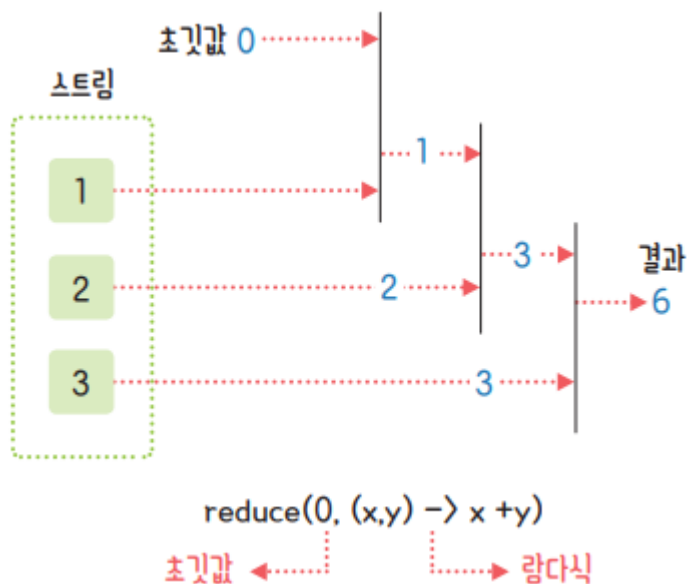
1

8

스트림을 이용한 집계와 수집

■ 리듀싱 연산

- 원소 개수, 합계, 평균값 등과 같은 단순 집계기가 아니라면 자바 API가 제공하는 기본 집계 연산으로 감당할 수 없다.
- 이를 위하여 Stream 인터페이스는 람다식을 사용하여 복합적인 집계 결과를 도출할 수 있도록 리듀싱 기능을 제공한다.
- 리듀싱 연산은 스트림원소를 단 하나의 값으로 축약시키는 연산이다.



스트림을 이용한 집계와 수집

■ 리듀싱 연산

- Stream과 IntStream의 리듀싱 연산(IntStream은 파란색)

메서드	의미
Optional reduce(BinaryOperator) OptionalInt reduce(IntBinaryOperator)	2개 원소를 조합해서 1개 값으로 축약하는 과정을 반복하여 집계한 결과를 반환한다.
T reduce(T, BinaryOperator) int reduce(int, IntBinaryOperator)	첫 번째 인숫값을 초깃값으로 제공한다는 것을 제외하면 위 메서드와 같다.

스트림을 이용한 집계와 수집

- 예제 : [sec05/Reduce1Demo](#),

```
import java.util.List;
import java.util.Optional;
import java.util.OptionalInt;

public class Reduce1Demo {
    public static void main(String[] args) {
        List<Integer> numbers = List.of(3, 4, 5, 1, 2);

        int sum1 = numbers.stream().reduce(0, (a, b) -> a + b);

        int sum2 = numbers.stream().reduce(0, Integer::sum);

        int mul1 = numbers.stream().reduce(1, (a, b) -> a * b);

        System.out.println(sum1);
        System.out.println(sum2);
        System.out.println(mul1);

        Optional<Integer> sum3 = numbers.stream().reduce(Integer::sum);

        OptionalInt sum4 = numbers.stream().mapToInt(x -> x.intValue()).reduce(Integer::sum);

        Optional<Integer> mul2 = numbers.stream().reduce((a, b) -> a * b);

        System.out.println(sum3.get());
        System.out.println(sum4.getAsInt());
        mul2.ifPresent(Util::print);
    }
}
```

15
15
120
15
15
120

스트림을 이용한 집계와 수집

■ 리듀싱 연산

- 예제 : [sec05/Reduce2Demo](#)

```
import java.util.DoubleSummaryStatistics;
import java.util.stream.DoubleStream;
import java.util.stream.Stream;

public class Reduce2Demo {
    public static void main(String[] args) {
        Stream<Nation> s1 = Nation.nations.stream();
        s1.reduce((n1, n2) -> n1.getPopulation() > n2.getPopulation() ? n1 : n2)
            .ifPresent(System.out::println);

        Stream<Nation> s2 = Nation.nations.stream();
        double sumOfPopulation = s2.filter(n -> n.getGdpRank() <= 20)
            .mapToDouble(n -> n.getPopulation())
            .reduce(0.0, (n1, n2) -> n1 + n2);
        System.out.println("리스트에서 GDP가 20위 이내의 나라의 인구 총합은 " +
            sumOfPopulation + "백만명이다.");

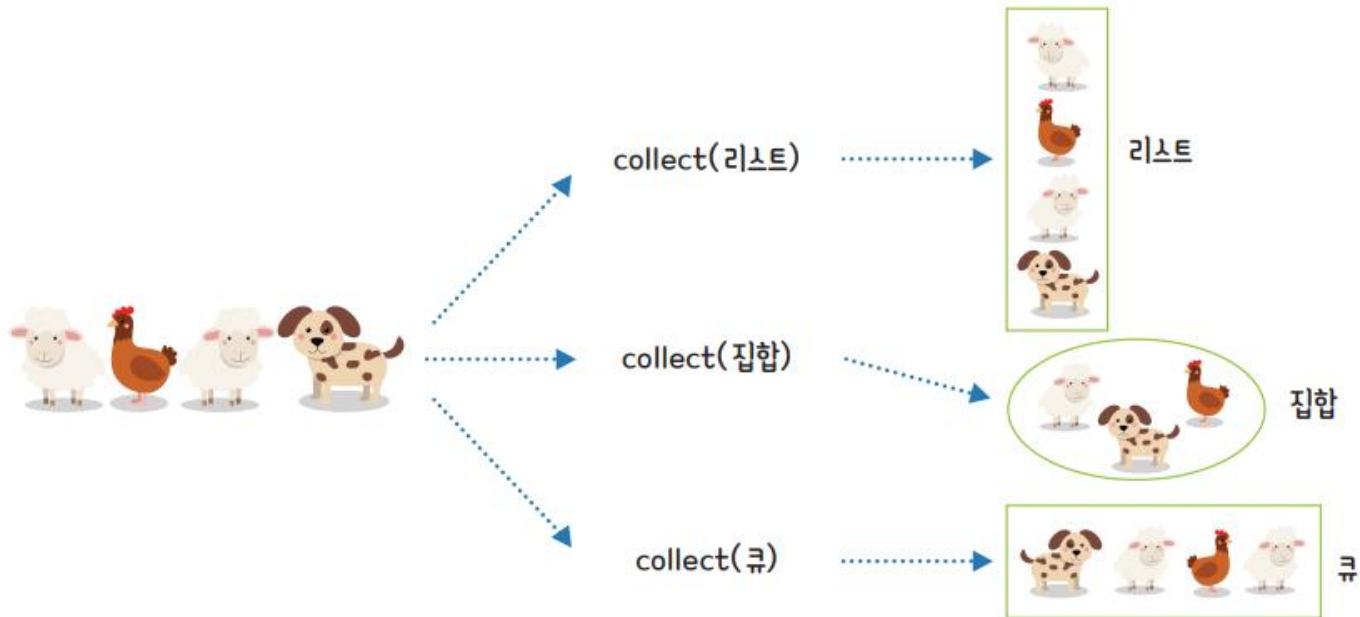
        Stream<Nation> s3 = Nation.nations.stream();
        DoubleStream ds = s3.mapToDouble(Nation::getPopulation);
        DoubleSummaryStatistics dss = ds.summaryStatistics();
        System.out.println(dss);
    }
}
```

China
리스트에서 GDP가 20위 이내의 나라의 인구 총합은 1789.7백만명이다.
DoubleSummaryStatistics{count=8, sum=1956.800000, min=4.500000, average=244.600000, max=1355.700000}

스트림을 이용한 집계와 수집

■ 컬렉터 연산

- 컬렉터는 원소를 어떤 컬렉션에 수집할 것인지 결정하며 다음과 같은 역할을 수행한다.
 - 원소를 컬렉션이나 StringBuilder와 같은 컨테이너에 수집한다.
 - 원소를 구분자와 같은 문자와 합칠 수 있다.
 - 원소를 그룹핑하여 새로운 컬렉션을 구성한다.



스트림을 이용한 집계와 수집

■ 컬렉터 연산 : 수집 및 요약

- Collectors 클래스가 제공하는 주요 정적 메서드

메서드	의미
Collector averagingInt(ToIntFunction)	정수 원소에 대한 평균을 도출한다.
Collector counting()	원소 개수를 헤아린다.
Collector joining(CharSequence)	문자열 원소를 연결하여 하나의 문자열로 결합한다.
Collector mapping(Function, Collector)	원소를 매핑한 후 컬렉터로 수집한다.
Collector maxBy(Comparator)	원소의 최댓값을 구한다.
Collector minBy(Comparator)	원소의 최솟값을 구한다.
Collector summingInt(ToIntFunction)	원소의 숫자 필드의 합계, 평균 등을 요약한다.
Collector toList()	원소를 List에 저장한다.
Collector toSet()	원소를 Set에 저장한다.
Collector toCollection(Supplier)	원소를 Supplier가 제공한 컬렉션에 저장한다.
Collector toMap(Function, Function)	원소를 키-값으로 Map에 저장한다.

스트림을 이용한 집계와 수집

- 예제 : [sec05/Collect1Demo](#)

```
import java.util.IntSummaryStatistics;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```
public class Collect1Demo {
    public static void main(String[] args) {
        Stream<Nation> sn = Nation.nations.stream();
        Double avg = sn.collect(Collectors.averagingDouble(Nation::getPopulation));
        System.out.println("인구 평균 : " + avg);

        sn = Nation.nations.stream();
        Long num = sn.collect(Collectors.counting());
        System.out.println("나라 개수 : " + num);

        sn = Nation.nations.stream();
        String name1 = sn.limit(4).map(Nation::getName).collect(Collectors.joining("-"));
        System.out.println("4개 나라(방법 1) : " + name1);

        sn = Nation.nations.stream();
        String name2 = sn.limit(4).collect(Collectors.mapping(Nation::getName, Collectors.joining("+")));
        System.out.println("4개 나라(방법 2) : " + name2);

        sn = Nation.nations.stream();
        Optional<Double> max = sn.map(Nation::getPopulation).collect(Collectors.maxBy(Double::compare));
        System.out.println("최대 인구 나라의 인구 수 : " + max);

        sn = Nation.nations.stream();
        IntSummaryStatistics sta = sn.collect(Collectors.summarizingInt(x -> x.getGdpRank()));
        System.out.println(sta);
    }
}
```

인구 평균 : 244.6

나라 개수 : 8

4개 나라(방법 1) : ROK-New Zealand-USA-China

4개 나라(방법 2) : ROK+New Zealand+USA+China

최대 인구 나라의 인구 수 : Optional[1355.7]

IntSummaryStatistics{count=8, sum=227, min=1, average=28.375000, max=63}

스트림을 이용한 집계와 수집

■ 컬렉터 연산 : 수집 및 요약

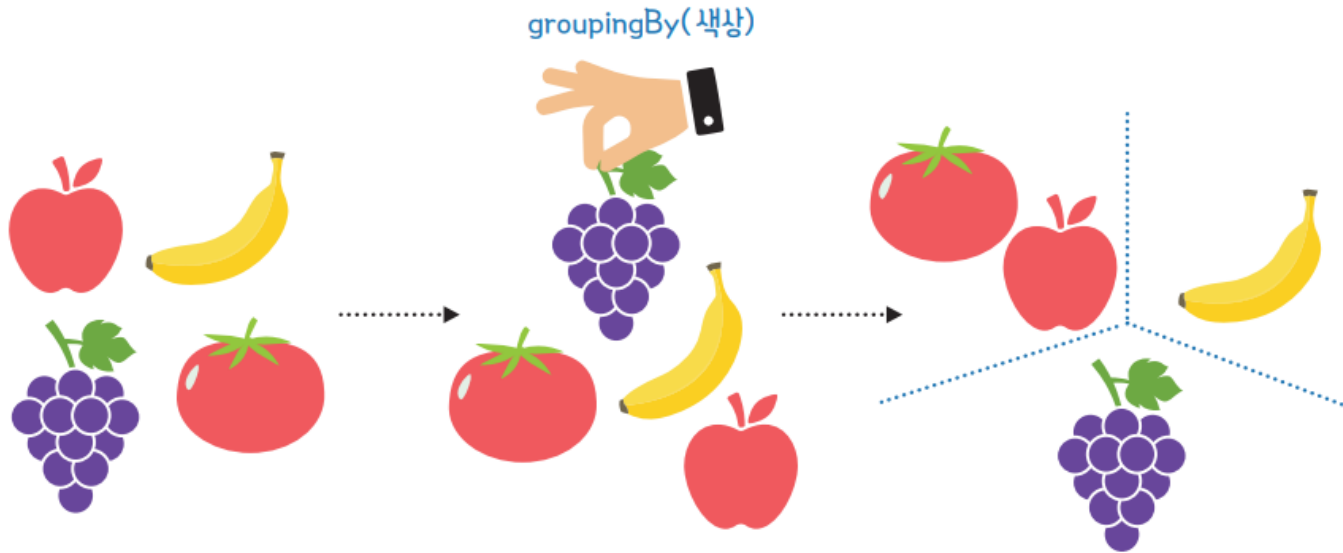
- 예제 : [sec05/Collect2Demo](#)

```
public class Collect2Demo {  
    public static void main(String[] args) {  
        Stream<Nation> sn = Nation.nations.stream();  
        Stream<String> ss = sn.map(Nation::getName).limit(3);  
        List<String> list = ss  
            .collect(Collectors.toList());  
        System.out.println(list);  
  
        sn = Nation.nations.stream();  
        Set<String> set = sn  
            .map(Nation::getName)  
            .limit(3)  
            .collect(Collectors.toSet());  
        System.out.println(set);  
  
        sn = Nation.nations.stream();  
        Map<String, Double> map = sn  
            .filter(Nation::isIsland)  
            .collect(Collectors.toMap(Nation::getName,  
                Nation::getPopulation));  
        System.out.println(map);  
  
        sn = Nation.nations.stream();  
        Set<Nation> hashSet = sn  
            .filter(Predicate.not(Nation::isIsland))  
            .collect(Collectors.toCollection(HashSet<Nation>::new));  
        hashSet.forEach(x ->  
            Util.print("(" + x.getName() + ", " + x.getGdpRank() + ")"));  
        System.out.println();  
    }  
}
```

[ROK, New Zealand, USA]
[New Zealand, USA, ROK]
{New Zealand=4.5, Sri Lanka=21.9, United Kingdom=63.7, Philiphine=107.7}
(USA, 1) (China, 2) (Morocco, 60) (ROK, 11)

스트림을 이용한 집계와 수집

- 스트림 원소를 그룹핑하여 새로운 컬렉션 구성
- 그룹핑
 - 키와 값의 쌍으로 이루어진 map 원소를 수집
 - 두번째 매개 값인 컬렉터가 없으면 list 타입
 - groupingBy() 연산으로 수집된 map 타입을 결과로 도출



```
Collector groupingBy(Function)  
Collector groupingBy(Function, Collector)
```

스트림을 이용한 집계와 수집

■ 그룹핑

- 예제 : [sec05/GroupingDemo](#)

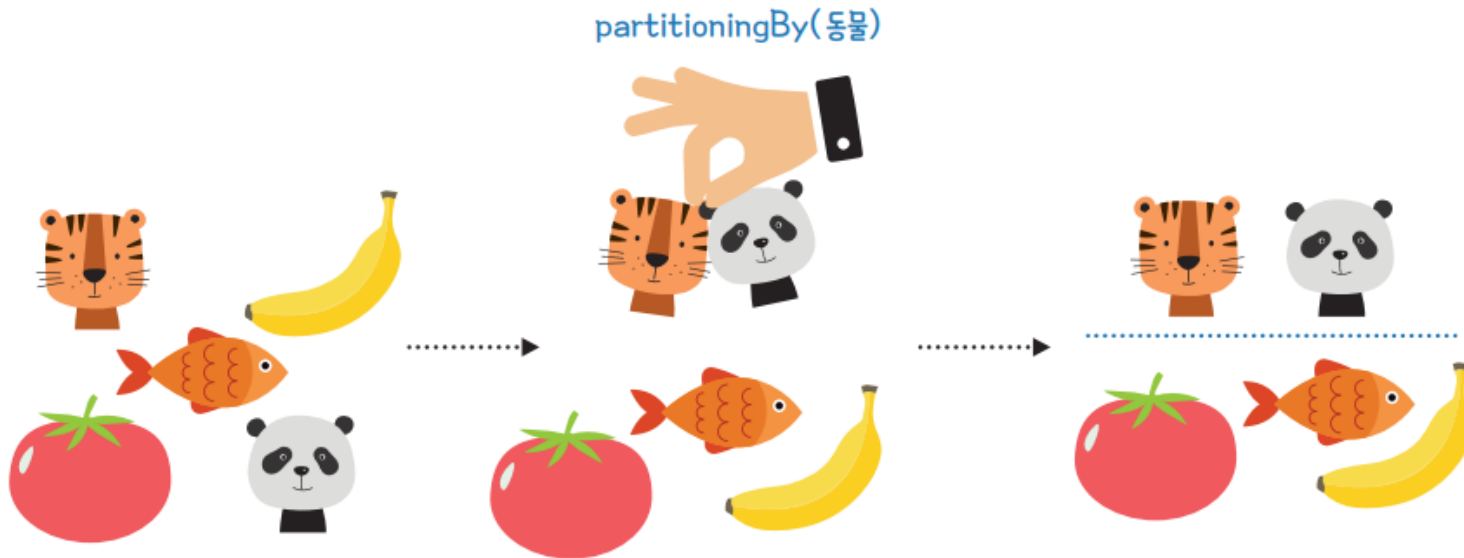
```
{LAND=[ROK, USA, China], ISLAND=[New Zealand]}  
{LAND=3, ISLAND=1}  
{LAND=ROK#USA#China, ISLAND=New Zealand}
```

```
import java.util.List;  
import java.util.Map;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class GroupingDemo {  
    public static void main(String[] args) {  
        Stream<Nation> sn = Nation.nations.stream().limit(4);  
        Map<Nation.Type, List<Nation>> m1 = sn  
            .collect(Collectors.groupingBy(Nation::getType));  
        System.out.println(m1);  
  
        sn = Nation.nations.stream().limit(4);  
        Map<Nation.Type, Long> m2 = sn  
            .collect(Collectors.groupingBy(Nation::getType, Collectors.counting()));  
        System.out.println(m2);  
  
        sn = Nation.nations.stream().limit(4);  
        Map<Nation.Type, String> m3 = sn.collect(  
            Collectors.groupingBy(Nation::getType,  
                Collectors.mapping(Nation::getName,  
                    Collectors.joining("#"))));  
        System.out.println(m3);  
    }  
}
```

스트림을 이용한 집계와 수집

■ 파티셔닝

- 조건에 따라 그룹핑



```
Collector partitioningBy(Predicate)  
Collector partitioningBy(Predicate, Collector)
```


스트림을 이용한 집계와 수집

■ 파티셔닝

● 예제 : [sec05/PartitioningDemo](#)

```
{false=[New Zealand], true=[ROK, USA, China]}  
{false=1, true=3}  
{false=New Zealand, true=ROK#USA#China}
```

```
import java.util.List;  
import java.util.Map;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
public class PartitioningDemo {  
    public static void main(String[] args) {  
        Stream<Nation> sn = Nation.nations.stream().limit(4);  
        Map<Boolean, List<Nation>> m1 = sn  
            .collect(Collectors.partitioningBy  
                (p -> p.getType() == Nation.Type.LAND));  
        System.out.println(m1);  
  
        sn = Nation.nations.stream().limit(4);  
        Map<Boolean, Long> m2 = sn  
            .collect(Collectors.partitioningBy(  
                p -> p.getType() == Nation.Type.LAND,  
                Collectors.counting()));  
        System.out.println(m2);  
  
        sn = Nation.nations.stream().limit(4);  
        Map<Boolean, String> m3 = sn  
            .collect(Collectors.partitioningBy(  
                p -> p.getType() == Nation.Type.LAND,  
                Collectors.mapping(Nation::getName,  
                    Collectors.joining("#"))));  
        System.out.println(m3);  
    }  
}
```