

## 17장. 스트림과 병렬처리

# Contents

- ❖ 1장. 스트림 소개
- ❖ 2장. 스트림 종류
- ❖ 3장. 스트림 파이프라인
- ❖ 4장. 필터링
- ❖ 5장. 매핑
- ❖ 6장. 정렬
- ❖ 7장. 루핑
- ❖ 8장. 매칭
- ❖ 9장. 기본 집계
- ❖ 10장. 커스텀 집계
- ❖ 11장. 수집
- ❖ 12장. 병렬 처리

# 1절. 스트림 소개

## ❖ 스트림이란?

- 자바 8부터 추가된 컬렉션(배열 포함)의 저장 요소를 하나씩 참조
- 람다식 (함수적-스타일(functional-style ) )으로 처리할 수 있도록 해주는 반복자

## ❖ 반복자 스트림

```
List<String> list = Arrays.asList("홍길동", "신용권", "감자바");
Iterator<String> iterator = list.iterator();
while(iterator.hasNext()) {
    String name = iterator.next();
    System.out.println(name);
}
```

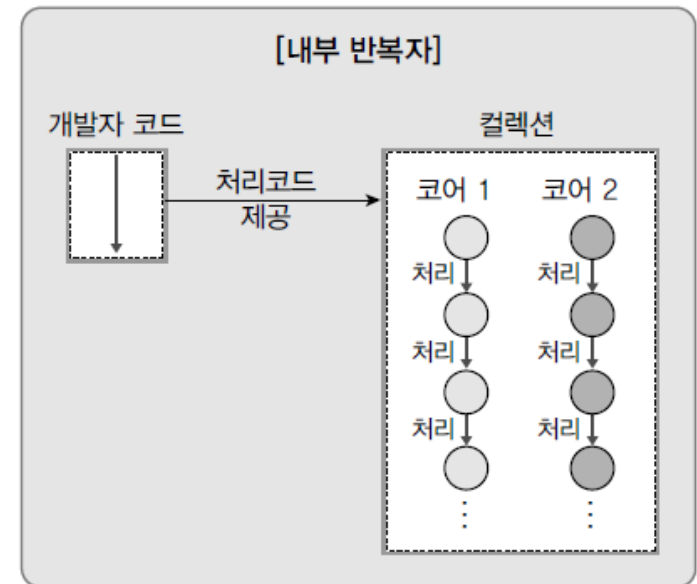


```
List<String> list = Arrays.asList("홍길동", "신용권", "감자바");
Stream<String> stream = list.stream();
stream.forEach( name -> System.out.println(name) );
```

# 1절. 스트림 소개

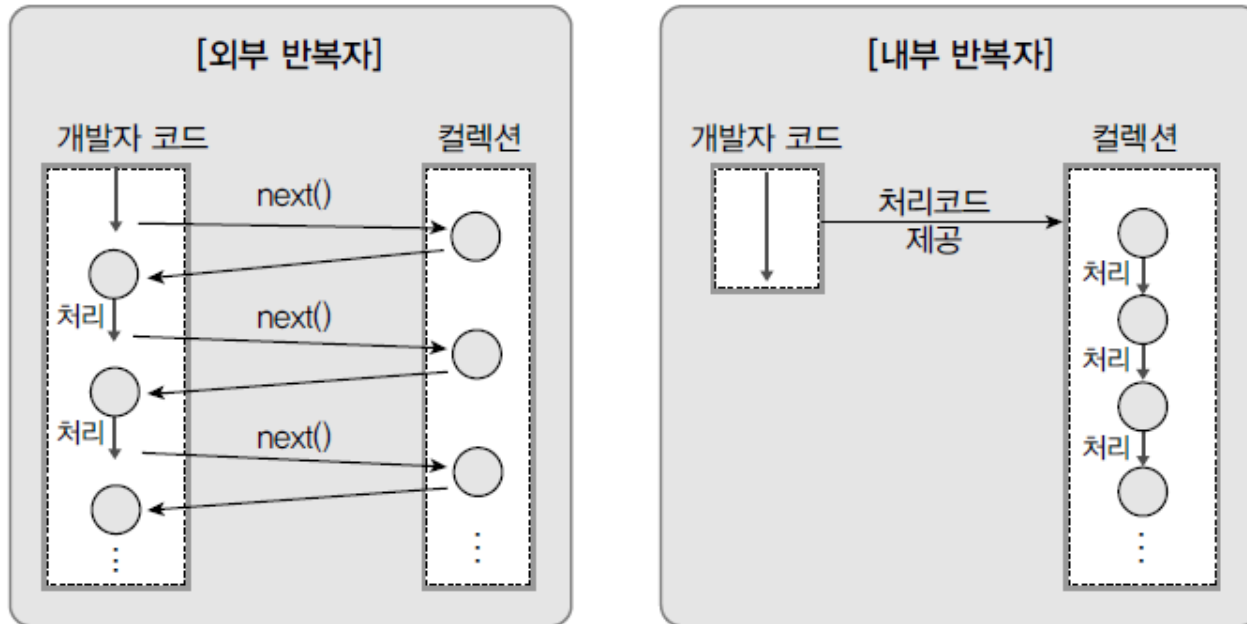
## ❖ 스트림의 특징

- Iterator와 비슷한 역할 하는 반복자
- 랴다식으로 요소 처리 코드 제공
  - 대부분의 요소처리 메소드는 함수적 인터페이스 매개 타입
- 내부 반복자 사용하므로 병렬 처리 쉬움
  - 컬렉션 내부에서 요소들 반복 시킴
  - 개발자는 요소당 처리해야 할 코드만 제공



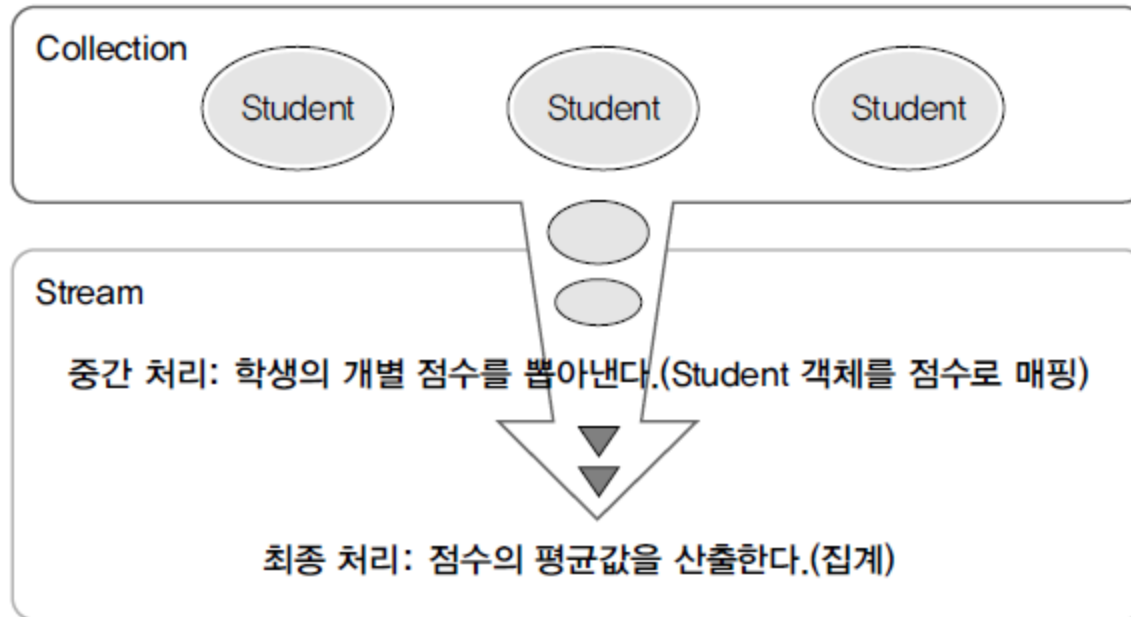
# 1절. 스트림 소개

## ■ 내부 반복자와 외부 반복자의 비교



# 1절. 스트림 소개

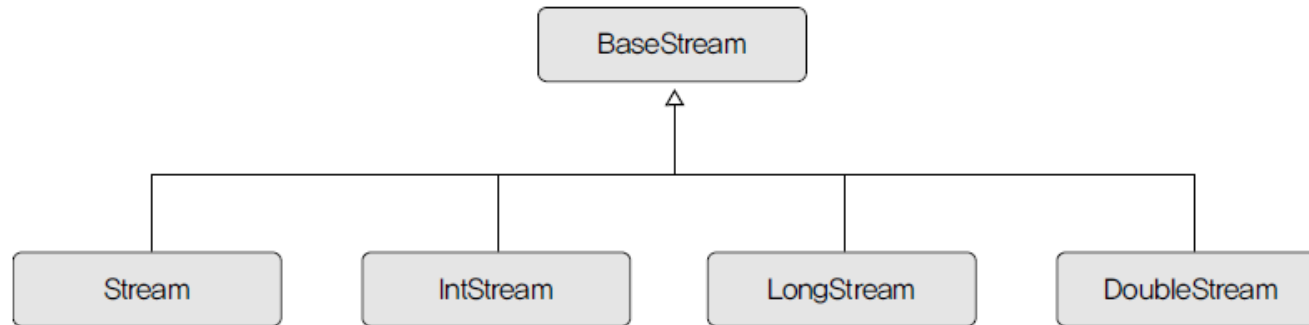
## ■ 스트림은 중간 처리와 최종 처리가 가능



## 2절. 스트림의 종류

### ❖ 스트림의 종류

- 자바 8부터 새로 추가
- `java.util.stream` 패키지에 스트림(stream) API 존재



- **BaseStream**
  - 모든 스트림에서 사용 가능한 공통 메소드 (직접 사용 X)
- **Stream** – 객체 요소 처리
- **IntStream, LongStream, DoubleStream**은 각각 기본 타입인 `int`, `long`, `double` 요소 처리

## 2절. 스트림의 종류

### ■ 스트림 인터페이스의 구현 객체 (p. 790~792)

- 주로 컬렉션과 배열에서 얻음
- 소스로부터 스트림 구현 객체 얻는 경우

리턴 타입	메소드(매개 변수)	소스
Stream<T>	java.util.Collection.stream() java.util.Collection.parallelStream()	컬렉션
Stream<T> IntStream LongStream DoubleStream	Arrays.stream(T[] ), Stream.of(T[] ) Arrays.stream(int[] ), IntStream.of(int[] ) Arrays.stream(long[] ), LongStream.of(long[] ) Arrays.stream(double[] ), DoubleStream.of(double[] )	배열
IntStream	IntStream.range(int, int) IntStream.rangeClosed(int, int)	int 범위
LongStream	LongStream.range(long, long) LongStream.rangeClosed(long, long)	long 범위
Stream<Path>	Files.find(Path, int, BiPredicate, FileVisitOption) Files.list(Path)	디렉토리
Stream<String>	Files.lines(Path, Charset) BufferedReader.lines()	파일
DoubleStream IntStream LongStream	Random.doubles(...) Random.ints() Random.longs()	랜덤 수



## 2절. 스트림의 종류

### ❖ 파일로부터 스트림 얻기

- Files의 정적 메소드인 lines ( )와 BufferedReader의 lines ( ) 메소드 이용
- 문자 파일 내용을 스트림 통해 행 단위로 읽고 콘솔에 출력
  - 예제 793p.

### ❖ 디렉토리로부터 스트림 얻기

- Files의 정적 메소드인 list ( ) 이용
- 디렉토리 내용(서브 디렉토리 또는 파일 목록)을 스트림 통해 읽고 콘솔에 출력

### 3절. 스트림 파이프라인

#### ❖ 리덕션(Reduction)

- 대량의 데이터를 가공해 축소하는 것
- 데이터의 합계, 평균값, 카운팅, 최대값, 최소값
- 컬렉션의 요소를 리덕션의 결과물로 바로 집계할 수 없을 경우에는?
  - 집계하기 좋도록 필터링, 매핑, 정렬, 그룹핑 등의 중간 처리 필요  
→ 스트림 파이프 라인의 필요성

#### ❖ 파이프라인

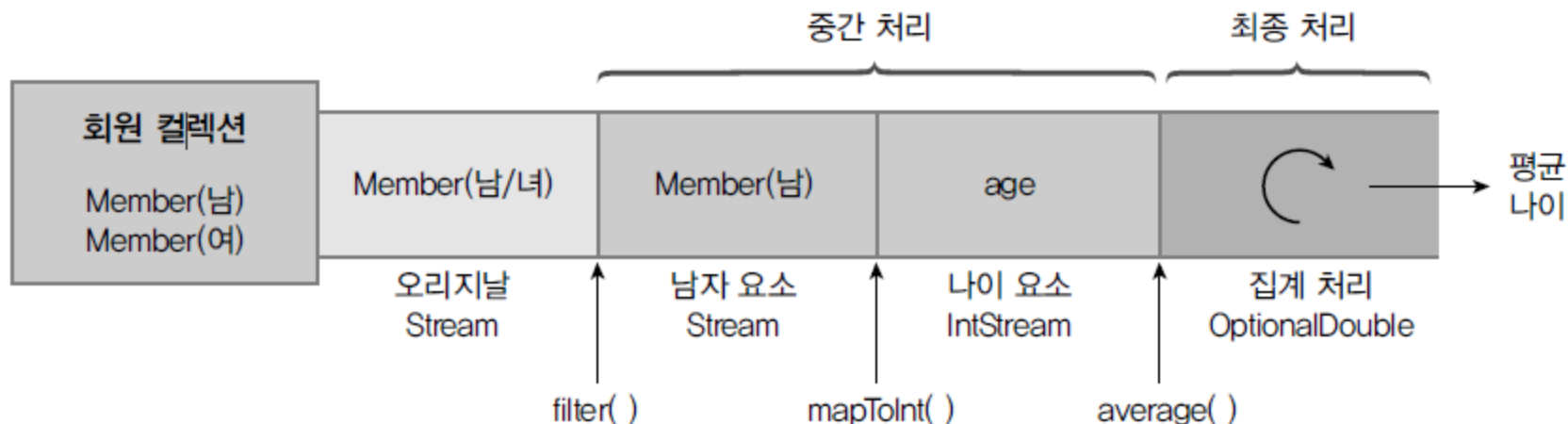
- 여러 개의 스트림이 연결되어 있는 구조
- 파이프라인에서 최종 처리를 제외하고는 모두 중간 처리 스트림



### 3절. 스트림 파이프라인

#### ❖ 중간 처리와 최종 처리

- Stream 인터페이스는 필터링, 매핑, 정렬 등의 많은 중간 처리 메소드 가짐
- 메소드들은 중간 처리된 스트림 리턴
- 스트림에서 다시 중간 처리 메소드 호출해 파이프라인 형성
- Ex) 회원들 중 남자 회원들의 나이 평균 구하기 (p.795~797)



# 3절. 스트림 파이프라인

## ❖ 중간 처리 메소드와 최종 처리 메소드

- 스트림이 제공하는 중간 처리용 메소드 – 리턴 타입이 스트림

종류	리턴 타입	메소드(매개 변수)	소속된 인터페이스
중간 처리	필터링	distinct()	공통
		filter(...)	공통
	매핑	flatMap(...)	공통
		flatMapToDouble(...)	Stream
		flatMapToInt(...)	Stream
		flatMapToLong(...)	Stream
		map(...)	공통
		mapToDouble(...)	Stream, IntStream, LongStream
		mapToInt(...)	Stream, LongStream, DoubleStream
		mapToLong(...)	Stream, IntStream, DoubleStream
		mapToObj(...)	IntStream, LongStream, DoubleStream
		asDoubleStream()	IntStream, LongStream
		asLongStream()	IntStream
		boxed()	IntStream, LongStream, DoubleStream
	정렬	sorted(...)	공통
	루핑	peek(...)	공통

# 3절. 스트림 파이프라인

## ■ 스트림이 제공하는 최종 처리용 메소드

- 리턴 타입이 기본 타입이거나 OptionalXXX

종류		리턴 타입	메소드(매개 변수)	소속된 인터페이스
최종 처리	매칭	boolean	allMatch(...)	공통
		boolean	anyMatch(...)	공통
		boolean	noneMatch(...)	공통
	집계	long	count( )	공통
		OptionalXXX	findFirst( )	공통
		OptionalXXX	max(...)	공통
		OptionalXXX	min(...)	공통
		OptionalDouble	average( )	IntStream, LongStream, DoubleStream
		OptionalXXX	reduce(...)	공통
		int, long, double	sum( )	IntStream, LongStream, DoubleStream
	루핑	void	forEach(...)	공통
	수집	R	collect(...)	공통

## 4절. 필터링

### ❖ 필터링이란?

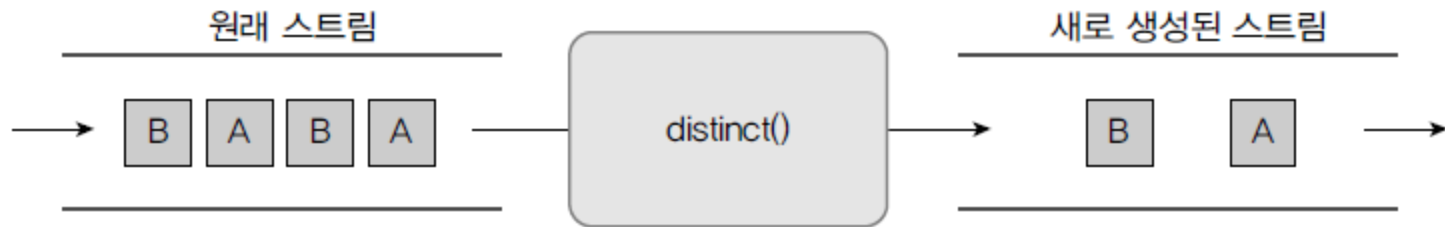
- 중간 처리 기능으로 요소 걸러내는 역할
- 필터링 메소드인 `distinct ( )`와 `filter ( )`메소드
  - 모든 스트림이 가지고 있는 공통 메소드

리턴 타입	메소드(매개 변수)	설명
Stream IntStream LongStream DoubleStream	<code>distinct( )</code>	중복 제거
	<code>filter(Predicate)</code>	조건 필터링
	<code>filter(IntPredicate)</code>	
	<code>filter(LongPredicate)</code>	
	<code>filter(DoublePredicate)</code>	

## 4절. 필터링

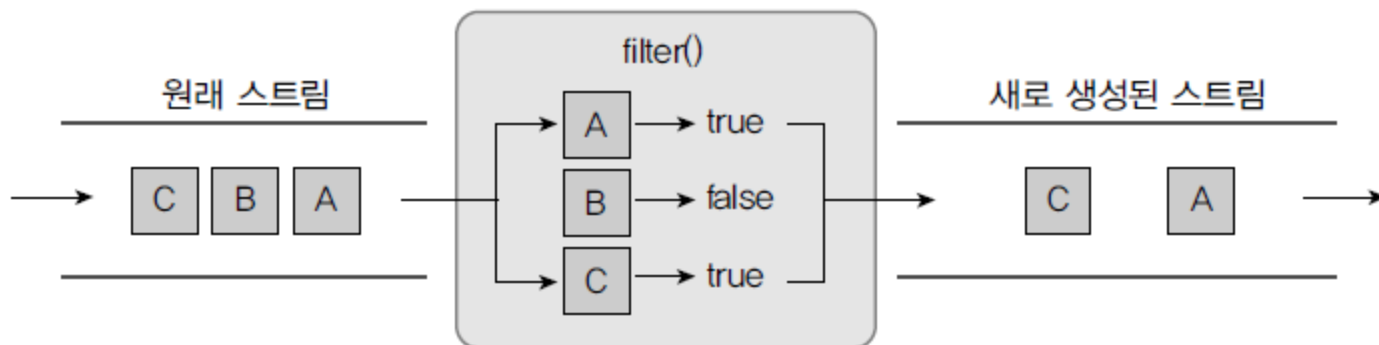
### ■ distinct ( ) 메소드 – 중복을 제거하는 기능

- Stream의 경우 Object.equals (Object)가 true
  - 동일한 객체로 판단해 중복 제거
- IntStream, LongStream, DoubleStream은 동일값일 경우 중복 제거



### ■ filter ( ) 메소드

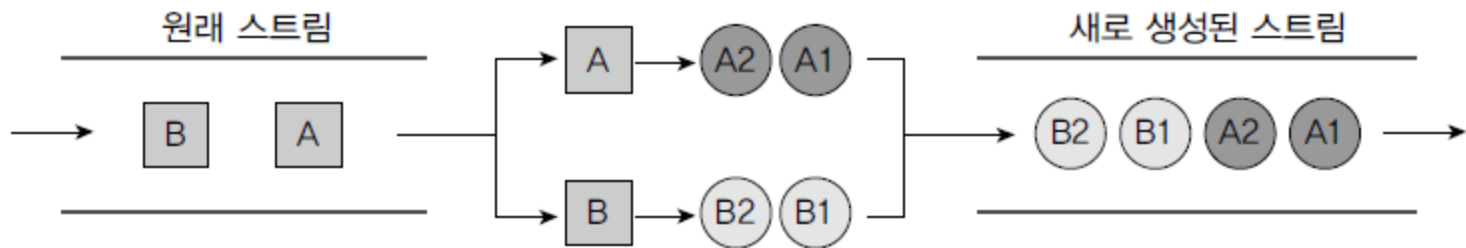
- 매개값으로 주어진 Predicate가 true를 리턴하는 요소만 필터링



## 5절. 매핑

### ❖ 매핑(mapping)

- 중간 처리 기능으로 스트림의 요소를 다른 요소로 대체하는 작업
- flatMapXXX( ) 메소드
  - 요소를 대체하는 복수 개의 요소들로 구성된 새로운 스트림 리턴



#### • flatMapXXX( ) 메소드의 종류

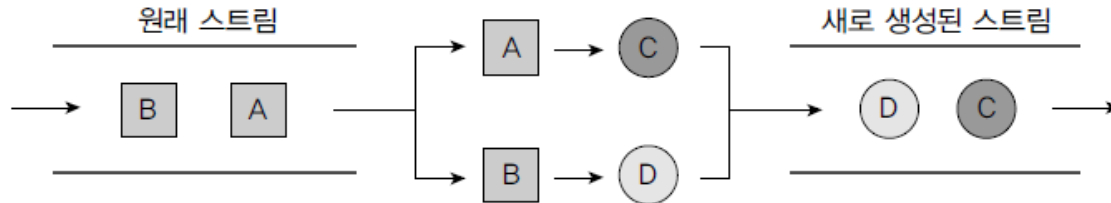
리턴 타입	메소드(매개 변수)	요소 → 대체 요소
Stream<R>	flatMap(Function<T, Stream< R>>)	T → Stream<R>
DoubleStream	flatMap(DoubleFunction<DoubleStream>)	double → DoubleStream
IntStream	flatMap(IntFunction<IntStream>)	int → IntStream
LongStream	flatMap(LongFunction<LongStream>)	long → LongStream
DoubleStream	flatMapToDouble(Function<T, DoubleStream>)	T → DoubleStream
IntStream	flatMapToInt(Function<T, IntStream>)	T → IntStream
LongStream	flatMapToLong(Function<T, LongStream>)	T → LongStream



# 5절. 매핑

## ■ mapXXX( ) 메소드

- 요소를 대체하는 요소로 구성된 새로운 스트림 리턴



리턴 타입	메소드(매개 변수)	요소 → 대체 요소
Stream<R>	map(Function<T, R>)	T → R
DoubleStream	mapToDouble(ToDoubleFunction<T>)	T → double
IntStream	mapToInt(ToIntFunction<T>)	T → int
LongStream	mapToLong(ToLongFunction<T>)	T → long
DoubleStream	map(DoubleUnaryOperator)	double → double
IntStream	mapToInt(DoubleToIntFunction)	double → int
LongStream	mapToLong(DoubleToLongFunction)	double → long
Stream<U>	mapToObj(DoubleFunction<U>)	double → U
IntStream	map(IntUnaryOperator mapper)	int → int
DoubleStream	mapToDouble(IntToDoubleFunction)	int → double
LongStream	mapToLong(IntToLongFunction mapper)	int → long
Stream<U>	mapToObj(IntFunction<U>)	int → U
LongStream	map(LongUnaryOperator)	long → long
DobleStream	mapToDouble(LongToDoubleFunction)	long → double
IntStream	mapToInt(LongToIntFunction)	long → Int
Stream<U>	mapToObj(LongFunction<U>)	long → U

## 5절. 매핑

### ❖ asDoubleStream(), asLongStream(), boxed() 메소드

리턴 타입	메소드(매개 변수)	설명
DoubleStream	asDoubleStream()	int → double long → double
LongStream	asLongStream()	int → long
Stream<Integer> Stream<Long> Stream<Double>	boxed()	int → Integer long → Long double → Double

## 6절. 정렬

### ❖ 정렬

- 스트림은 요소가 최종 처리되기 전에 중간 단계에서 요소를 정렬
- 최종 처리 순서 변경 가능
- 요소를 정렬하는 메소드

리턴 타입	메소드(매개 변수)	설명
Stream<T>	sorted()	객체를 Comparable 구현 방법에 따라 정렬
Stream<T>	sorted(Comparator<T>)	객체를 주어진 Comparator에 따라 정렬
DoubleStream	sorted()	double 요소를 오름차순으로 정렬
IntStream	sorted()	int 요소를 오름차순으로 정렬
LongStream	sorted()	long 요소를 오름차순으로 정렬

## 7절. 루핑

### ❖ 루핑(looping)

- 요소 전체를 반복하는 것
- peek( )
  - 중간 처리 메소드
  - 중간 처리 단계에서 전체 요소를 루핑하며 추가 작업 하기 위해 사용
  - 최종처리 메소드가 실행되지 않으면 지연
    - 반드시 최종 처리 메소드가 호출되어야 동작
- forEach( )
  - 최종 처리 메소드
  - 파이프라인 마지막에 루핑하며 요소를 하나씩 처리
  - 요소를 소비하는 최종 처리 메소드
    - sum( )과 같은 다른 최종 메소드 호출 불가

## 8절. 매칭

### ❖ 매칭이란?

- 최종 처리 단계에서 요소들이 특정 조건에 만족하는지 조사하는 것
- allMatch ( ) 메소드
  - 모든 요소들이 매개값으로 주어진 Predicate의 조건을 만족하는지 조사
- anyMatch( ) 메소드
  - 최소한 한 개의 요소가 매개값으로 주어진 Predicate 조건을 만족하는지 조사
- noneMatch( ) 메소드
  - 모든 요소들이 매개값으로 주어진 Predicate의 조건을 만족하지 않는지 조사

## 9절. 기본 집계

### ❖ 집계(Aggregate)

- 최종 처리 기능으로 요소들을 처리해 카운팅, 합계, 평균값, 최대값, 최소값 등과 같이 하나의 값으로 산출하는 것
- 집계는 대량의 데이터를 가공해서 축소하는 리덕션 (Reduction)
- 스트림이 제공하는 기본 집계

리턴 타입	메소드(매개 변수)	설명
long	count()	요소 개수
OptionalXXX	findFirst()	첫 번째 요소
Optional<T> OptionalXXX	max(Comparator<T>) max()	최대 요소
Optional<T> OptionalXXX	min(Comparator<T>) min()	최소 요소
OptionalDouble	average()	요소 평균
int, long, double	sum()	요소 총합

## 9절. 기본 집계

### ❖ Optional 클래스

- Optional, OptionalDouble, OptionalInt, OptionalLong 클래스
- 저장하는 값의 타입만 다를 뿐 제공하는 기능은 거의 동일
- 단순히 집계 값만 저장하는 것이 아님
- 집계 값이 존재하지 않을 경우 디폴트 값을 설정 가능
- 집계 값을 처리하는 Consumer도 등록 가능
- Optional 클래스들이 제공하는 메소드들

리턴 타입	메소드(매개 변수)	설명
boolean	isPresent()	값이 저장되어 있는지 여부
T double int long	orElse(T) orElse(double) orElse(int) orElse(long)	값이 저장되어 있지 않을 경우 디폴트 값 지정
void	ifPresent(Consumer) ifPresent(DoubleConsumer) ifPresent(IntConsumer) ifPresent(LongConsumer)	값이 저장되어 있을 경우 Consumer에서 처리

# 10절. 커스텀 집계

## ❖ 커스텀 집계를 위한 메소드

- `sum()`, `average()`, `count()`, `max()`, `min()` 이용
  - 기본 집계 메소드 이용
- `reduce()` 메소드
  - 프로그램화해서 다양한 집계 결과물 만들 수 있도록 제공

인터페이스	리턴 타입	메소드(매개 변수)
Stream	Optional<T>	<code>reduce(BinaryOperator&lt;T&gt; accumulator)</code>
	T	<code>reduce(T identity, BinaryOperator&lt;T&gt; accumulator)</code>
IntStream	OptionalInt	<code>reduce(IntBinaryOperator op)</code>
	int	<code>reduce(int identity, IntBinaryOperator op)</code>
LongStream	OptionalLong	<code>reduce(LongBinaryOperator op)</code>
	long	<code>reduce(long identity, LongBinaryOperator op)</code>
DoubleStream	OptionalDouble	<code>reduce(DoubleBinaryOperator op)</code>
	double	<code>reduce(double identity, DoubleBinaryOperator op)</code>



# 11절. 수집

## ❖ 수집 기능

- 요소들을 필터링 또는 매핑 한 후 요소들을 수집하는 최종 처리 메소드 인 `collect ( )`
- 필요한 요소만 컬렉션으로 받을 수 있음
- 요소들을 그룹핑 한 후 집계(리덕션)

## ❖ 필터링한 요소 수집

- `Stream`의 `collect (Collector<T,A,R> collector )` 메소드
  - 필터링 또는 매핑된 요소들을 새로운 컬렉션에 수집하고, 이 컬렉션 리턴

리턴 타입	메소드(매개 변수)	인터페이스
R	<code>collect(Collector&lt;T,A,R&gt; collector)</code>	Stream

# 11절. 수집

## ❖ 매개값인 Collector (수집기 p.819~822)

- 어떤 요소를 어떤 컬렉션에 수집할 것인지 결정
- Collector의 타입 파라미터 T는 요소
- A는 누적기(accumulator)
- R은 요소가 저장될 컬렉션
  - 해석하면 T 요소를 A 누적기가 R에 저장한다는 의미
- Collectors 클래스의 정적 메소드

리턴 타입	Collectors의 정적 메소드	설명
Collector<T, ?, List<T>>	toList()	T를 List에 저장
Collector<T, ?, Set<T>>	toSet()	T를 Set에 저장
Collector<T, ?, Collection<T>>	toCollection( Supplier<Collection<T>> )	T를 Supplier가 제공한 Collection에 저장
Collector<T, ?, Map<K,U>>	toMap( Function<T,K> keyMapper, Function<T,U> valueMapper)	T를 K와 U로 매핑해서 K를 키로, U를 값으로 Map에 저장
Collector<T, ?, ConcurrentMap<K,U>>	toConcurrentMap( Function<T,K> keyMapper, Function<T,U> valueMapper)	T를 K와 U로 매핑해서 K를 키로, U를 값으로 ConcurrentMap에 저장

# 11절. 수집

## ❖ 사용자 정의 컨테이너에 수집하기 (p.823~826)

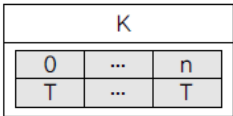
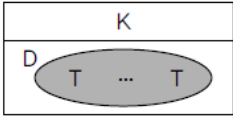
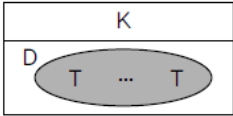
- 스트림은 요소들을 필터링, 또는 매핑해 사용자 정의 컨테이너 객체에 수집할 수 있도록
- 이를 위한 추가적인 `collect ( )` 메소드

인터페이스	리턴 타입	메소드(매개 변수)
Stream	R	<code>collect(Supplier&lt;R&gt;, BiConsumer&lt;R,? super T&gt;, BiConsumer&lt;R,R&gt;)</code>
IntStream	R	<code>collect(Supplier&lt;R&gt;, ObjIntConsumer&lt;R&gt;, BiConsumer&lt;R,R&gt;)</code>
LongStream	R	<code>collect(Supplier&lt;R&gt;, ObjLongConsumer&lt;R&gt;, BiConsumer&lt;R,R&gt;)</code>
DoubleStream	R	<code>collect(Supplier&lt;R&gt;, ObjDoubleConsumer&lt;R&gt;, BiConsumer&lt;R,R&gt;)</code>

# 11절. 수집

## ❖ 요소를 그룹핑해서 수집 (p.826~830)

- collect ( )를 호출시 Collectors의 groupingBy( ) 또는 groupingByConcurrent( )가 리턴하는 Collector를 매개값 대입
  - 컬렉션의 요소들을 그룹핑해서 Map객체 생성

리턴 타입	Collectors의 정적 메소드	설명
Collector<T,?,Map<K,List<T>>>	groupingBy(Function<T, K> classifier)	T를 K로 매핑하고 K키에 저장된 List에 T를 저장한 Map 생성
Collector<T,?, ConcurrentMap<K,List<T>>>	groupingByConcurrent(Function<T,K> classifier)	
Collector<T,?,Map<K,D>>	groupingBy(Function<T, K> classifier, Collector<T,A,D> collector)	T를 K로 매핑하고 K키에 저장된 D객체에 T를 누적한 Map 생성
Collector<T,?, ConcurrentMap<K,D>>	groupingByConcurrent(Function<T,K> classifier, Collector<T,A,D> collector)	
Collector<T,?,Map<K,D>>	groupingBy(Function<T,K> classifier, Supplier<Map<K,D>> mapFactory, Collector<T,A,D> collector)	T를 K로 매핑하고 Supplier가 제공하는 Map에서 K키에 저장된 D객체에 T를 누적
Collector<T,?, ConcurrentMap<K,D>>	groupingByConcurrent(Function<T,K> classifier, Supplier<ConcurrentMap<K,D>> mapFactory, Collector<T,A,D> collector)	

# 11절. 수집

## ❖ 그룹핑 후 매핑 및 집계 (p.831~834)

### ■ `Collectors.groupingBy()` 메소드

- 그룹핑 후, 매핑이나 집계(평균, 카운팅, 연결, 최대, 최소, 합계)를 할 수 있도록 두 번째 매개값으로 `Collector`를 가질 수 있는 특성

리턴 타입	메소드(매개 변수)	설명
<code>Collector&lt;T,?,R&gt;</code>	<code>mapping( Function&lt;T, U&gt; mapper, Collector&lt;U,A,R&gt; collector)</code>	T를 U로 매핑한 후, U를 R에 수집
<code>Collector&lt;T,?,Double&gt;</code>	<code>averagingDouble( ToDoubleFunction&lt;T&gt; mapper)</code>	T를 Double로 매핑한 후, Double의 평균값을 산출
<code>Collector&lt;T,?,Long&gt;</code>	<code>counting()</code>	T의 카운팅 수를 산출
<code>Collector &lt;CharSequence,?,String&gt;</code>	<code>joining(CharSequence delimiter)</code>	<code>CharSequence</code> 를 구분자 (delimiter)로 연결한 String을 산출
<code>Collector&lt;T,?,Optional&lt;T&gt;&gt;</code>	<code>maxBy( Comparator&lt;T&gt; comparator)</code>	<code>Comparator</code> 를 이용해서 최대 T를 산출
<code>Collector&lt;T,?,Optional&lt;T&gt;&gt;</code>	<code>minBy( Comparator&lt;T&gt; comparator)</code>	<code>Comparator</code> 를 이용해서 최소 T를 산출
<code>Collector&lt;T,?,Integer&gt;</code>	<code>summingInt(ToIntFunction) summingLong(ToLongFunction) summingDouble(ToDoubleFunction)</code>	Int, Long, Double 타입의 합계 산출

# 12절. 병렬 처리

## ❖ 병렬 처리(Parallel Operation)

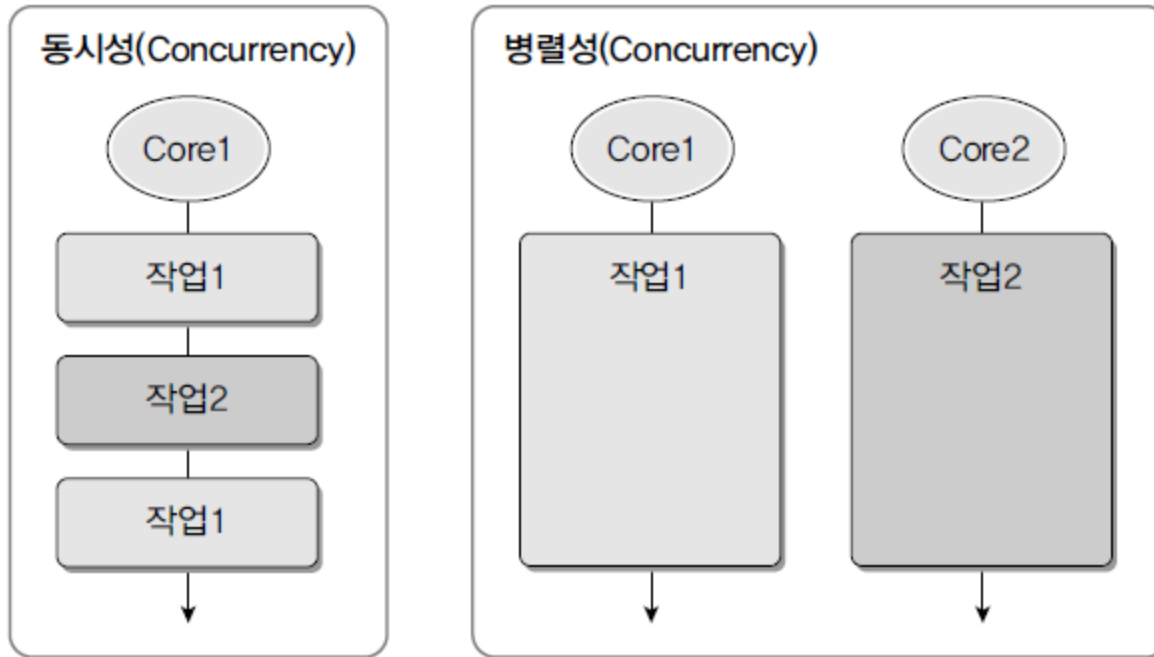
- 멀티 코어 CPU 환경에서 쓰임
- 하나의 작업을 분할해서 각각의 코어가 병렬적 처리하는 것
- 병렬 처리의 목적은 작업 처리 시간을 줄이기 위한 것
- 자바 8부터 요소를 병렬 처리할 수 있도록 하기 위해 병렬 스트림 제공

## ❖ 동시성(Concurrency)과 병렬성(Parallelism)

- 동시성 - 멀티 작업을 위해 멀티 스레드가 번갈아 가며 실행하는 성질
  - 싱글 코어 CPU를 이용한 멀티 작업
    - 병렬적으로 실행되는 것처럼 보임
    - 실체는 번갈아 가며 실행하는 동시성 작업
- 병렬성 - 멀티 작업 위해 멀티 코어 이용해서 동시에 실행하는 성질

# 12절. 병렬 처리

## ❖ 동시성과 병렬성의 비교



# 12절. 병렬 처리

## ❖ 병렬성의 종류

### ■ 데이터 병렬성

- 전체 데이터를 쪼개어 서브 데이터들로 만든 뒤 병렬 처리해 작업을 빨리 끝내는 것
- 자바 8에서 지원하는 병렬 스트림은 데이터 병렬성을 구현한 것
- 멀티 코어의 수만큼 대용량 요소를 서브 요소들로 나누고
- 각각의 서브 요소들을 분리된 스레드에서 병렬 처리
  - ex) 쿼드 코어(Quad Core) CPU일 경우 4개의 서브 요소들로 나누고, 4개의 스레드가 각각의 서브 요소들을 병렬 처리



# 12절. 병렬 처리

## ■ 작업 병렬성

- 작업 병렬성은 서로 다른 작업을 병렬 처리하는 것
- Ex) 웹 서버 (Web Server )
  - 각각의 브라우저에서 요청한 내용을 개별 스레드에서 병렬로 처리

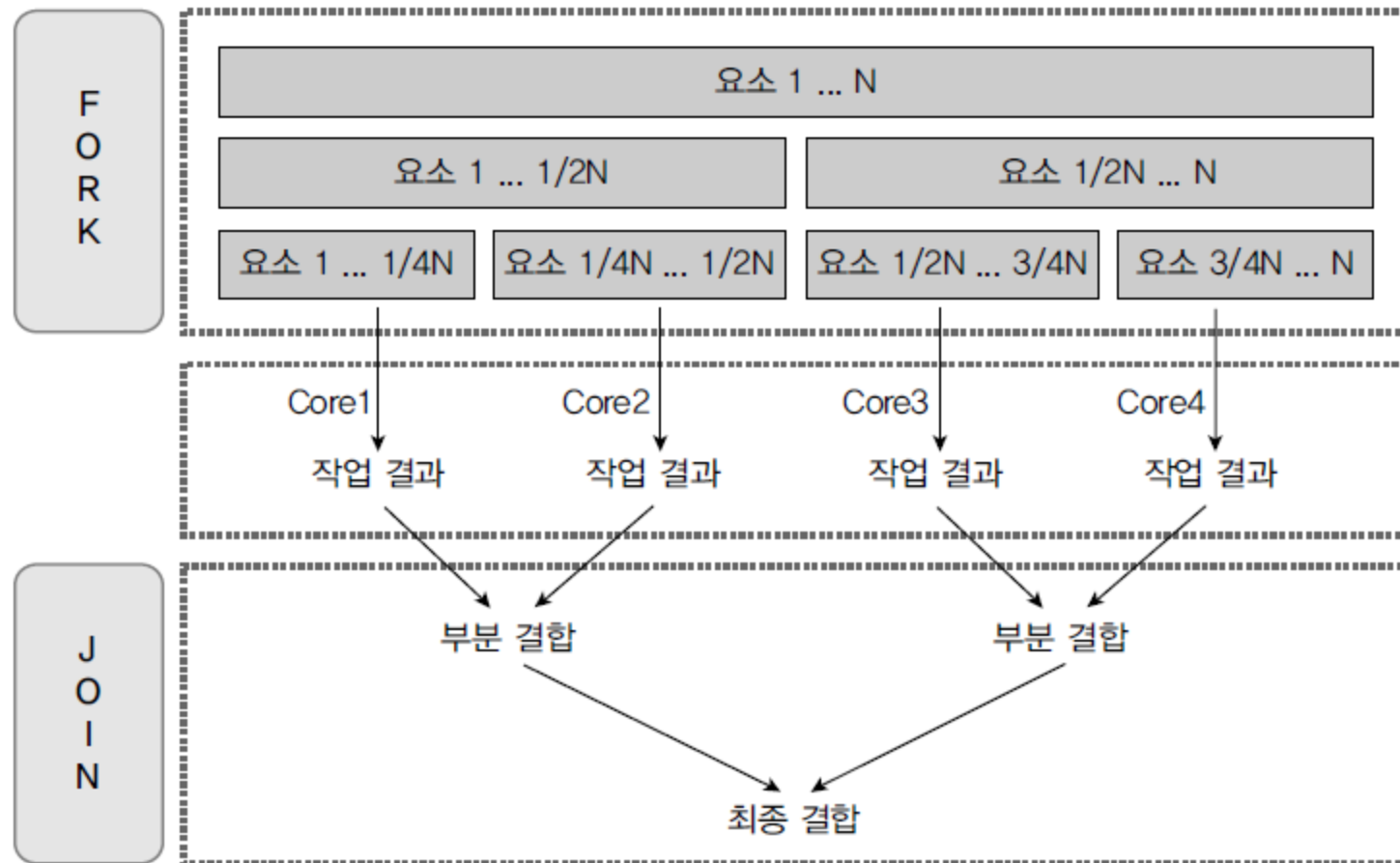
## ❖ 포크 조인 프레임워크

- 런타임 시 포크조인 프레임워크 동작
- 포크 단계에서는 전체 데이터를 서브 데이터로 분리
- 서브 데이터를 멀티 코어에서 병렬로 처리
- 조인 단계에서는 서브 결과를 결합해서 최종 결과 도출

# 12절. 병렬 처리

## ❖ 포크 조인 프레임 워크의 원리

### ■ ForkJoinPool 사용해 작업 스레드 관리



# 12절. 병렬 처리

## ❖ 병렬 스트림 생성

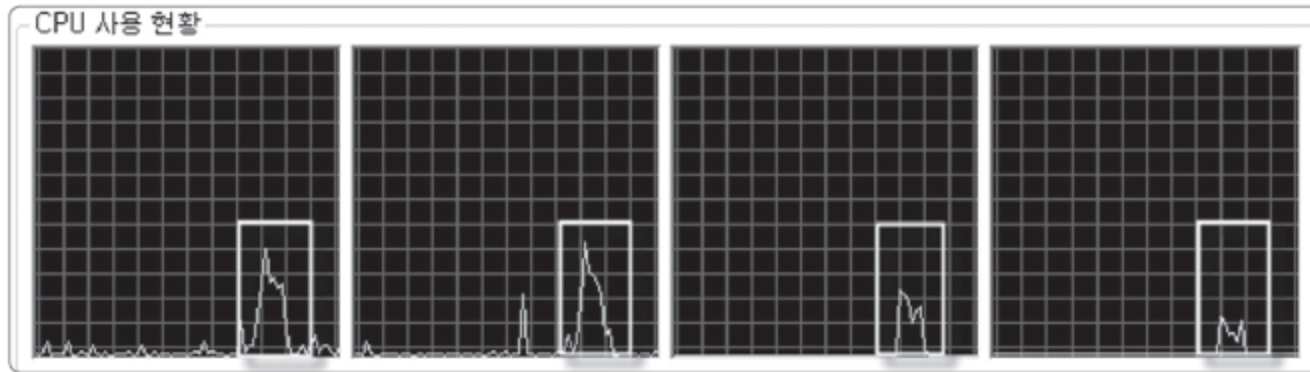
- 코드에서 포크조인 프레임워크 사용해도 병렬처리 가능
- 병렬 스트림 이용할 경우 백그라운드에서 포크조인 프레임 워크 동작
  - 매우 쉽게 구현해 사용 가능
- 병렬 스트림을 얻는 메소드
  - `parallelStream ( )` 메소드
    - 컬렉션으로부터 병렬 스트림을 바로 리턴
  - `parallel ( )` 메소드
    - 순차 처리 스트림을 병렬 처리 스트림으로 변환해서 리턴

인터페이스	리턴 타입	메소드(매개 변수)
<code>java.util.Collection</code>	<code>Stream</code>	<code>parallelStream( )</code>
<code>java.util.Stream.Stream</code>	<code>Stream</code>	<code>parallel( )</code>
<code>java.util.Stream.IntStream</code>	<code>IntStream</code>	
<code>java.util.Stream.LongStream</code>	<code>LongStream</code>	
<code>java.util.Stream.DoubleStream</code>	<code>DoubleStream</code>	

# 12절. 병렬 처리

## ❖ 병렬 처리가 정상적으로 이루어질 때 CPU의 상태

### ■ 쿼드 코어 CPU



## ❖ 병렬 처리 성능

### ■ 병렬 처리에 영향을 미치는 3가지 요인

- 요소의 수와 요소당 처리 시간
  - 요소 수가 적고 요소당 처리 시간 짧으면 순차 처리가 빠름
- 스트림 소스의 종류
  - ArrayList, 배열은 인덱스로 요소 관리 → 병렬처리가 빠름
- 코어(Core)의 수 – 싱글 코어의 경우 순차 처리가 빠름