스트림과
병렬 처리

01 스트림 소개

스트림은 자바8부터 추가된 컬렉션의 요소를 하나씩 참조해서 람다식으로 처리할 수 있도록 해주는 반복자이다.

자바 7 이전까지는 배열 또는 컬렉션 인스턴스를 다루기 위해서 for 또는 Foreach문을 돌면서 요소 하나씩을 다루는 방법이 있었다.

이러한 코드들을 스트림을 이용하여 함수 여러 개를 조합하여 원하는 결과를 필터링하고 가공된 결과를 얻을 수 있다. 또한 람다를 이용해서 코드의 양을 줄이고 간결하게 표현할 수 있다.

```java
//java 7이전
List<String> result1 = new ArrayList<>();
for(String str1 : list1){
    for(String str2 : list2){
        if(str1.equals(str2)){
            result1.add(str1);
        }
    }
}
for(String str : result1){
    System.out.print(str);
}
System.out.println();
```

```java
//스트림을 이용
List<String> result2 =
list1.stream()
        .filter(str -> list2.stream().anyMatch(Predicate.isEqual(str)))
        .collect(Collectors.toList());

result2.stream().forEach(System.out::print);
```

01 스트림의 특징

스트림은 Iterator와 비슷한 역할을 하는 반복자이지만,
람다식으로 요소 처리 코드를 제공하는 점과 내부 반복자를 사용하므로
병렬 처리가 쉽다는 점 그리고 중간 처리와 최종 처리 작업을 수행할 수 있다.

람다식으로 요소 처리 코드를 제공한다.

Stream이 제공하는 대부분의 요소 처리 메소드는 함수적 인터페이스 매개 타입을 가지기 때문에 람다식 또는 메소드 참조를 이용해서 요소 처리 내용을 매개값으로 전달할 수 있다.

내부 반복자를 사용하여 병렬 처리가 쉽다.

내부 반복자는 사용자가 반복당 수행할 액션만을 작성하여 제공하고, 그 액션을 컬렉션이 받아 내부적으로 처리하는 방식이다. 이때 수행 액션은 보통 콜백 함수로 전달된다. 그래서 개발자는 요소 처리 코드에만 집중 할 수 있다.

스트림은 중간 처리와 최종 처리를 할 수 있다.

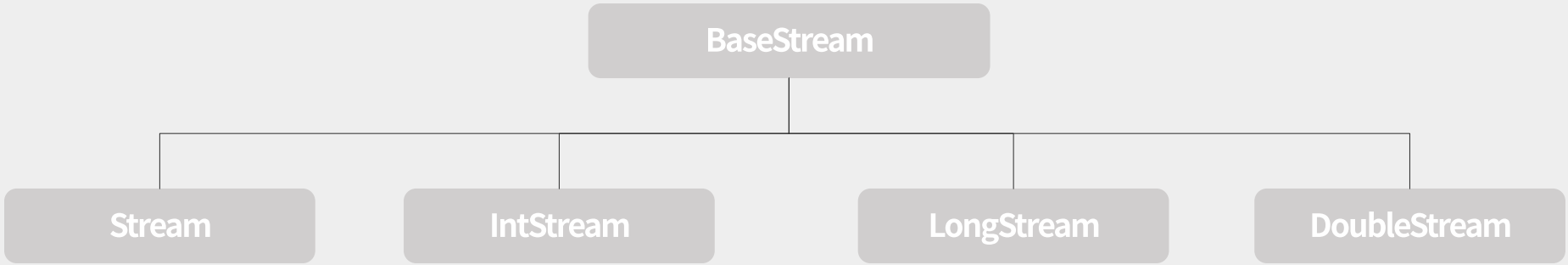
중간 처리에서는 매핑, 필터링, 정렬을 수행하고 최종 처리에서는 반복, 카운팅, 평균, 총합 등의 집계 처리를 수행한다.

02 스트림의 종류

Java.util.stream 패키지에는 스트림 API들이 포진하고 있다.

패키지 내용을 보면 BaseStream 인터페이스를 부모로 해서 자식 인터페이스들이
상속 관계를 이루고 있다.

BaseStream 인터페이스에는 공통 메소드들이 정의되어 있고 직접적으로 사용되진 않는다.



02 스트림의 종류

Java.util.stream 패키지에는 스트림 API들이 포진하고 있다.

패키지 내용을 보면 BaseStream 인터페이스를 부모로 해서 자식 인터페이스들이
상속 관계를 이루고 있다.

BaseStream 인터페이스에는 공통 메소드들이 정의되어 있고 직접적으로 사용되진 않는다.

리턴 타입	메소드(매개변수)	요소
Stream<T>	java.util.Collection.Stream() java.util.Collection.parallelStream()l	컬렉션
Stream<T> IntStream LongStream DoubleStream	Arrays.stream(T[]) Arrays.stream(int[]) Arrays.stream(long[]) Arrays.stream(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)	파일
DoubleStream IntStream LongStream	Random.doubles(...) Random.ints() Random.longs	랜덤 수

02 컬렉션으로부터 스트림 얻기

List<Student> 컬렉션에서 Stream<Student>를 얻어내고 요소를 콘솔에 출력하기

String[]과 int[]배열로부터 스트림을 얻어내고 콘솔에 출력하기

```
    List<Student> studentList = Arrays.asList(
            new Student( name: "홍길동",  score: 10),
            new Student( name: "신용권",  score: 20),
            new Student( name: "유미선",  score: 30)
    );

    Stream<Student> stream = studentList.stream();
    stream.forEach(s-> System.out.println("s.getName() = " + s.getName()));
}
```

```
class Student{
    private String name;
    private int score;

    public Student (String name, int score){
        this.name = name;
        this.score = score;
    }

    public String getName() {return name;}
    public int getScore() {return score;}
}
```

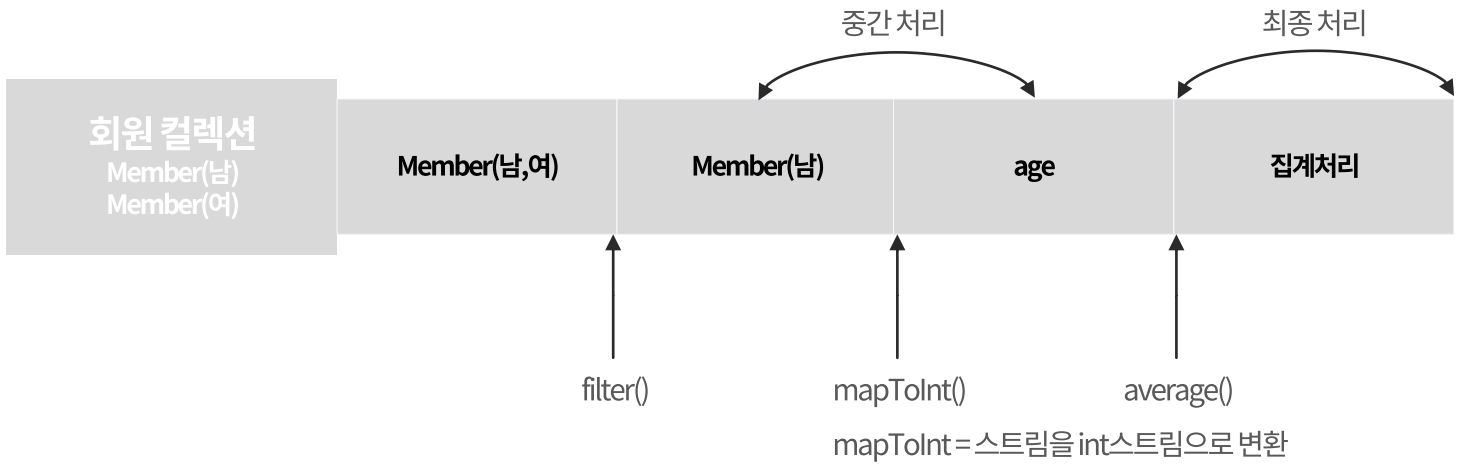
```
String[] strArray = {"홍길동", "신용권", "김미나"};
Stream<String> strStream = Arrays.stream(strArray);
strStream.forEach(a -> System.out.println(a + ", "));

int[] intArray = {1,2,3,4,5};
IntStream intStream = Arrays.stream(intArray);
intStream.forEach(a -> System.out.print(a + ", "));
```

```
홍길동,
신용권,
김미나,
1, 2, 3, 4, 5,
```

03 스트림 파이프라인

대량의 데이터를 가공해서 축소하는 것을 일반적으로 리덕션(Reduction)이라고 한다.

컬렉션의 요소를 리덕션의 결과물로 바로 집계할 수 없을 경우에는 집계하기 좋도록 필터링, 매핑, 정렬, 그룹핑 등의 중간 처리가 필요하다.



파이프라인은 여러 개의 스트림이 연결되어 있는 구조를 말한다. 파이프라인에서 최종 처리를 제외하고는 모두 중간 처리 스트림이다.

중간 스트림이 생성될 때 바로 중간 처리(필터링, 매핑, 정렬)되는 것이 아니라 최종 처리가 시작되기 전까지 중간 처리는 지연(lazy)된다. 최종 처리가 시작되면 비로소 컬렉션의 요소가 하나씩 중간 스트림에서 처리되고 최종 처리까지 오게 된다.

03 스트림 파이프라인

대량의 데이터를 가공해서 축소하는 것을 일반적으로 리덕션(Reduction)이라고 한다.

컬렉션의 요소를 리덕션의 결과물로 바로 집계할 수 없을 경우에는 집계하기 좋도록 필터링, 매핑, 정렬, 그룹핑 등의 중간 처리가 필요하다.

```
List<Member> list = Arrays.asList(
        new Member("홍길동", Member.Male, 30)
);
Stream<Member> maleFemaleStream = list.stream();
Stream<Member> maleStream =
        maleFemaleStream.filter(m -> m.getSex()==Member.MALE);
IntStream ageStream = maleStream.mapToInt(Member::getAge);
OptionalDouble optionalDouble = ageStream.average();
double ageAvg = optionalDouble.getAsDouble();
```

```
double ageAvg = list.stream() Stream<Member>
        .filter(m -> m.getSex() == Member.MALE)
        .mapToInt(Member :: getAge) IntStream
        .average() OptionalDouble
        .getAsDouble();
```

로컬 변수 생략

파이프라인은 여러 개의 스트림이 연결되어 있는 구조를 말한다. 파이프라인에서 최종 처리를 제외하고는 모두 중간 처리 스트림이다.

중간 스트림이 생성될 때 바로 중간 처리(필터링, 매핑, 정렬)되는 것이 아니라 최종 처리가 시작되기 전까지 중간 처리는 지연(lazy)된다. 최종 처리가 시작되면 비로소 컬렉션의 요소가 하나씩 중간 스트림에서 처리되고 최종 처리까지 오게 된다.

03 중간 처리 메소드와 최종 처리 메소드

다음은 스트림이 제공하는 중간 처리와 최종 처리를 하는 메소드를 설명한 표이다.

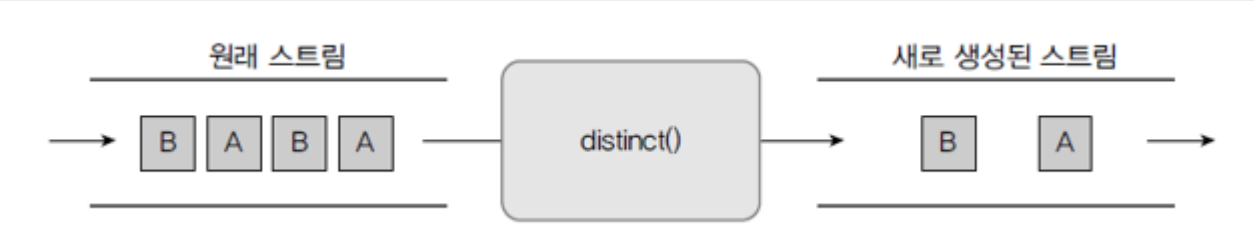
종류		리턴 타입	메소드(매개 변수)	소속된 인터페이스
중간 처리	필터링	Stream IntStream LongStream DoubleStream	distinct( )	공통
			filter(…)	공통
	매핑		flatMap(…)	공통
			flatMapToDouble(…)	Stream
			flatMapToInt(…)	Stream
			flatMapToLong(…)	Stream
			map(…)	공통
			mapToDouble(…)	Stream, IntStream, LongStream
			mapToInt(…)	Stream, LongStream,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

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

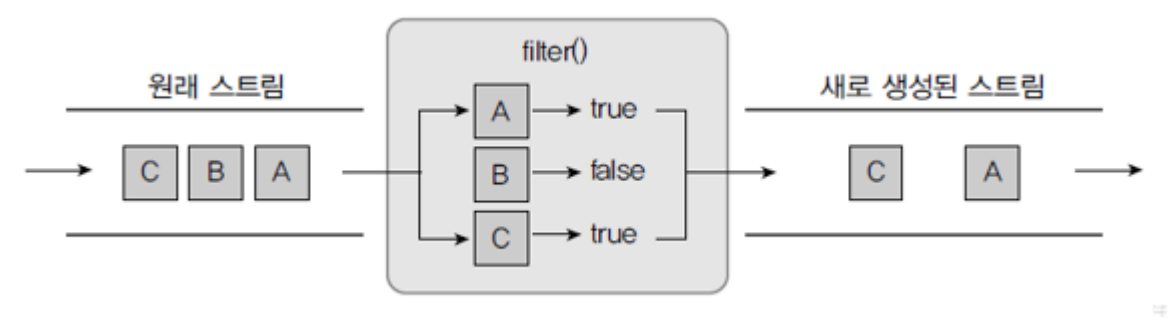
04 필터링-distinct(), filter()

필터링은 중간 처리 기능으로 요소를 걸러내는 역할을 한다.

필터링 메소드인 distinct, filter는 모든 스트림이 가지고 있는 공통 메소드이다.



Distinct 메소드는 중복을 제거한다.
스트림의 경우 Object.equals(Object)가 true이면 동일 객체로 판단하고 중복을 제거한다.



Filter 메소드는 매개값으로 주어진 Predicate가 true를 리턴하는 요소만 필터링한다.

05 매핑 – mapXXXX()

mapXXX() 메소드는 요소를 대체하는 요소로 구성된 새로운 스트림을 리턴한다.

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

05 매핑 – flatMapXXX()

매핑은 중간처리 기능으로 스트림의 요소를 다른 요소로 대체하는 작업을 말한다.

flatMapXXX()는 요소를 대체하는 복수 개의 요소들로 구성된 새로운 스트림을 리턴한다.

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 -> InputStream
LongStream	flatMapToLong(Function<T, LongStream>)	T -> LongStream

05 매핑 –asDoubleStream, asLongStream, boxed

asDoubleStream 메소드는 intStream의 int 요소 또는 longStream의 long 요소를 double 요소로 타입 변환해서 DoubleStream을 생성한다. asLongStream도 long요소로 타입변환 시킨다.

Boxed 메소드는 int, long, double 요소를 Integer, Long, Double 요소로 박싱해서 Stream을 생성한다.

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

06 정렬(sorted)

스트림은 요소가 최종 처리되기 전에 중간 단계에서 요소를 정렬하여 최종 처리 순서를 변경할 수 있다.

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

객체 요소일 경우에는 클래스에 Comparable을 구현해야 하고 구현한 요소에서만 sorted메소드를 호출해야 한다.

Comparable은 객체를 비교하게 해주는 인터페이스이고 Comparable을 사용하고자 한다면 compareTo 메소드를 반드시 재정의 해줘야 한다.

객체 요소가 comparable을 구현하지 않았다면 comparator를 매개값으로 갖는 sorted메소드를 사용하면 된다.

```java
public class Student implements Comparable<Student>{
    private String name;
    private int score;

    public Student(String name, int score) {
        this.name = name;
        this.score = score;
    }
    public String getName(){
        return name;
    }
    public int getScore(){
        return score;
    }
    /*
    @Override
    public int compareTo(Student o){
        return Integer.compare(score, o.score);
    }
    */
}
```

07 루핑(peek, forEach), 매칭(allMatch, anyMatch, noneMatch)

루핑은 요소 전체를 반복하는 것을 말한다. 루핑하는 메소드에는 peek와 forEach가 있다.

Peek는 중간처리 메소드이고 forEach는 최종 처리 메소드이다.

```
int[] intArr = {1,2,3,4,5};

System.out.println("peek를 마지막에 호출");
Arrays.stream(intArr)
        .filter(a -> a%2==0)
        .peek(n -> System.out.println(n)); //동작하지 않음
```

Peek는 중간 처리 단계에서 전체 요소를 루핑하면서 추가적인 작업을 하기위해 사용한다.
최종처리 메소드가 실행되지 않으면 지연되기 때문에 반드시 최종 처리 메소드가 호출되어야 동작한다.

```
Arrays.stream(intArr)
        .filter(a -> a%2==0)
        .forEach(n -> System.out.println(n));
```

forEach는 최종처리 메소드이기 때문에 파이프라인 마지막에 루핑하면서 요소를 하나씩 처리한다.

스트림 클래스는 최종 처리 단계에서 요소들이 특정 조건에 만족하는지 조사할 수 있도록 세 가지 매칭 메소드를 제공하고 있다.

allMatch : 모든 요소들이 매개값으로 주어진 Predicate의 조건을 만족하는지 조사한다.

anyMatch : 최소한 한 개의 요소가 매개값으로 주어진 Predicate의 조건을 만족하는지 조사한다.

noneMatch : 모든 요소들이 매개값으로 주어진 Predicate의 조건을 만족하지 않는지 조사한다.

08 집계 – Optional 클래스

집계는 최종 처리 기능으로 요소들을 처리해서 카운팅, 합계, 평균값, 최대값, 최소값 등과 같이 하나의 값으로 산출하는 것을 말한다.

집계는 대량의 데이터를 가공해서 축소하는 리덕션(Reduction)이라고 볼 수 있다.

Optional 클래스는 집계 값이 존재하지 않을 경우 디폴트 값을 설정할 수 있고, 집계 값을 처리하는 Consumer도 등록할 수 있다.

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

```
public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    double avg = list.stream() Stream<Integer>
            .mapToInt(Integer :: intValue) IntStream
            .average() OptionalDouble
            .getAsDouble();
    System.out.println("평균 : " + avg);
}
```

09 커스텀 집계

스트림은 기본 집계 메소드들을 제공하지만, 프로그램화해서 다양한 집계 결과물을 만들 수 있도록 reduce() 메소드도 제공한다.

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

위의 스트림 인터페이스에는 매개타입으로 …Operator, 리턴타입으로 Optional…, int, long, doubl을 가지는 reduce()메소드가 오버로딩 되어있다.

스트림 요소가 전혀 없을 경우 디폴트 값인 identity 매개값이 리턴된다.

Reduce(identify, accumulator)
초기값 누적 수행 작업

10 수집(collect)

스트림 API는 요소들을 필터링 또는 매핑한 후 요소들을
수집하는 최종 처리 메소드인 collect를 제공하고 있다.

스트림의 collect(Collector<T,A,R> collector) 메소드는 필터링 또는
매핑된 요소들을 새로운 컬렉션에 수집하고, 이 컬렉션을 리턴한다.

표에 설명되어있는 ConcurrentMap과 Map의 차이점은 Map은 스레드에 안전하지 않고,
ConcurrentMap은 스레드에 안전하다.

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

스트림은 요소들을 필터링, 또는 매핑해서 사용자 정의 컨테이너 객체에 수집할 수 있도록
Collect() 메소드를 추가적으로 제공한다.

리턴 타입	메소드(매개 변수)	인터페이스
R	collect(Collector<T,A,R> collector)	Stream

인터페이스	리턴 타입	메소드(파라미터)
Stream	R	collect(Supplier<R>, BiConsumer<R, ? super T>, Biconsumer<R, R>)
IntStream	R	collect(Supplier<R>, ObjIntConsumer<R>, Biconsumer<R, R>)
LongStream	R	collect(Supplier<R>, ObjLongConsumer<R>, Biconsumer<R, R>)
DoubleStream	R	collect(Supplier<R>, ObjDoubleConsumer<R>, Biconsumer<R, R>)

10 수집(collect)

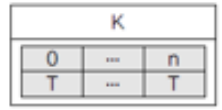
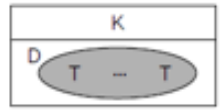
Collect 메소드는 단순히 요소를 수집하는 기능 이외에 컬렉션의 요소들을 그룹핑해서 Map객체를 생성하는 기능도 제공한다.

Collect()를 호출할 때 Collectors의 groupingBy() 또는 groupingByConcurrent()가 리턴하는 Collectors를 매개값으로 대입하면 된다.

Collectors.groupingBy() 메소드는 그룹핑 후, 매핑이나 집계를 할 수 있도록 두번째 매개값으로 Collector를 가질 수 있다.

Collectors는 mapping() 메소드 이외에도 집계를 위해 다양한 Collector를 리턴하는 메소드들을 제공하고 있다.

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

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)	

11 병렬 처리

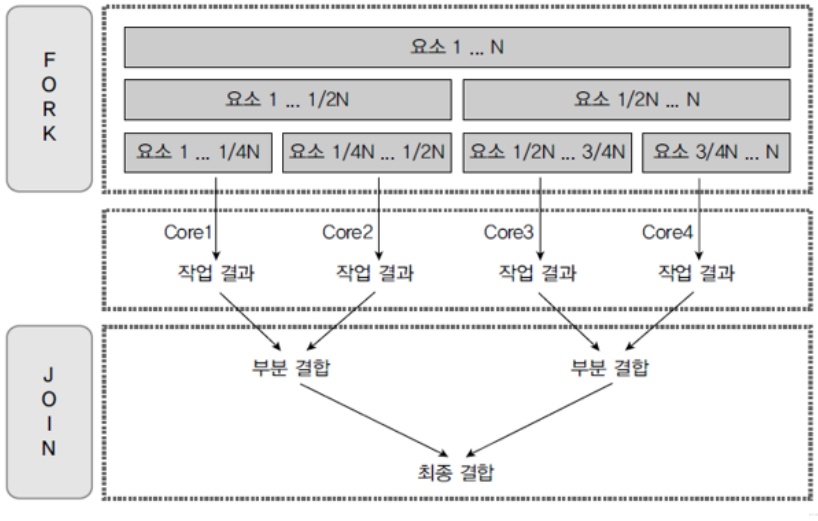
병렬 처리(Parallel Operation)란 멀티 코어 CPU 환경에서 하나의 작업을 분할해서 각각의 코어가 병렬적으로 처리하는 것을 말한다. (작업 시간을 줄이기 위해)

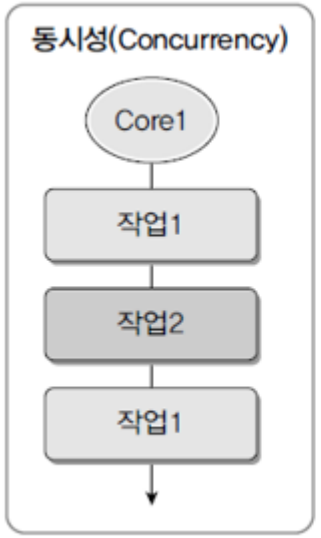
멀티 스레드는 동시성 또는 병렬성으로 실행되지만 동시성과 병렬성은 목적이 다르다.

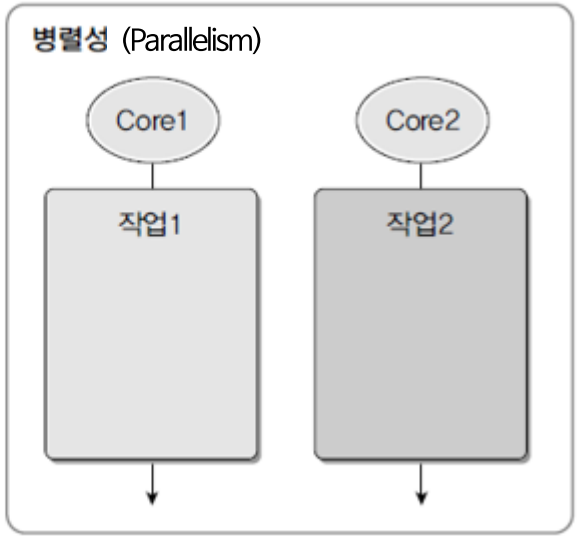
동시성 : 멀티 작업을 위해 멀티 스레드가 번갈아 가면서 실행하는 성질이다. 싱글코어 CPU를 이용한 멀티 작업은 병렬적으로 실행되는 것처럼 보이지만 실제로는 동시성 작업이다.

병렬성 : 병렬성은 멀티 작업을 위해 멀티 코어를 이용해 실행하는 성질이다.

병렬성은 데이터 병렬성(Data parallelism)과 작업 병렬성(Task parallism)으로 구분할 수 있다.







11 병렬 스트림 생성

병렬 스트림을 이용할 경우에는 백그라운드에서 포크조인 프레임워크가 사용되기 때문에 개발자는 아주 쉽게 병렬 처리를 할 수 있다.

병렬 스트림은 위의 parrallelStream, parallel 메소드로 얻을 수 있다.

parallelStream 메소드는 컬렉션으로부터 병렬 스트림을 바로 리턴한다.

Parallel메소드는 순차 처리 스트림을 병렬 처리 스트림으로 변환해서 리턴한다.

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

성능

병렬 스트림은 병렬화하기 위해 스트림을 재귀적으로 분할하고, 스레드를 할당하고,
최종적으로 부분 결과를 하나로 합치는 과정이 필요하기 때문에 오히려 속도가 느릴 수 있다.

- 요소의 수가 많고 요소당 처리시간이 긴 경우
- 스트림 소스의 종류
- 코어의 수
- 병렬로 수행하기 어려운 스트림 모델
- 박싱의 최소화
- 순서에 의존하는 연산

THANK YOU