Chapter 1

익명객체와 람다식

익명 객체 란

인터페이스 타입으로 변수를 선언하고, 구현 객체를 초기값으로 대입하는 경우를 생각해보자.
인터페이스의 구현 클래스를 선언하고,
new 연산자를 이용해서 구현 객체를 생성한 후, 필드나 로컬 변수에 대입하는 것이 기본이다.

하지만,
구현 클래스가 매번 달라지거나, 한 번만 사용되는 경우, 굳이 구현 클래스를 생성하지 않고
익명 클래스로(이름 없는 클래스) 로 선언 할 수 있습니다.

```
interface Car {
    public void run();
}

public class Garage {

    public Car car = ??;
}
```

```
interface Car {
    public void run();
}

public class Garage {

    public Car car = new Car() {

        @Override
        public void run() {
            System.out.println("달립니다");
        }
    };
}
```

멤버변수 Car에는 구현 클래스가 들어가야한다

람다식 이란

함수적 프로그래밍
Y = f(x); 형태의 함수로 구성된 프로그래밍 기법
함수를 매개 값으로 전달하고 결과를 받는 코드로 구성

함수적 프로그래밍이 객체 지향 프로그래밍 보다는 효율적인 경우.
1. 대용량 데이터의 처리시
2. 이벤트 지향 프로그램시

최근 프로그래밍 기법
객체지향프로그래밍 + 함수적 프로그래밍

자바 8부터 함수적 프로그래밍 지원
- 람다식(Lamda Expression)를 제공합니다
- 익명 함수를 생성하기 위한 식

(매개변수, 매개변수) -> { 실행문 }

람다식의 장점
- 코드가 간결해 진다.
- 컬렉션 요소 처리가 쉬워진다.

람다식의 기본 사용

자바의 람다식은 함수적 인터페이스의 익명 구현 객체를 대신합니다.

람다식(Lambda Expression)은 코드블럭(code block)을 메소드에 넣을 때 사용하는 기술입니다.

함수적 인터페이스란?

추상 메서드가 1개인 인터페이스

```
public interface Say01 {
     public void talking();
}
```

다음 greeting메서드를 실행시키려면???

어떻게 해야 하는가???

```
public class Person {
     public void greeting(Say01 say) {
          say.talking();
     }
}
```

방법1 – 익명 객체 방법

```
Person p = new Person();
p.greeting(new Say01() {

     public void talking() {
          System.out.println("안녕");
     }
});
```

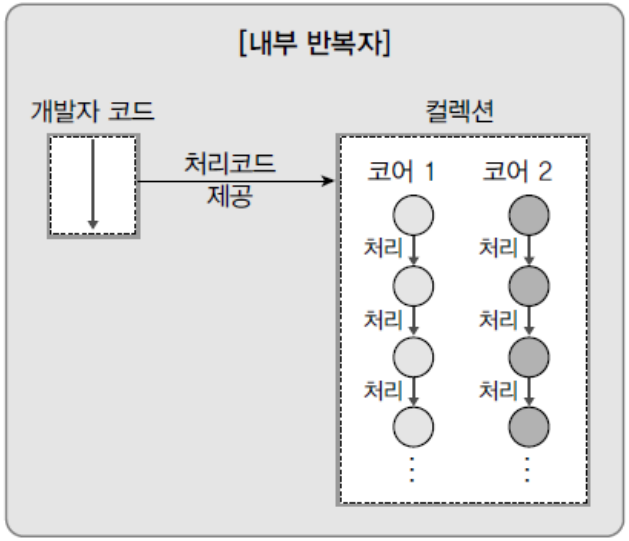
방법1 – 람다식

```
Person pp = new Person();
pp.greeting( () -> {
     System.out.println("안녕");
});
```

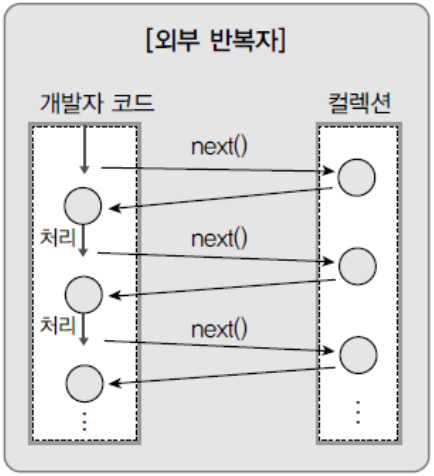
람다식을 적용하기 위한 스트림 소개

반복자 스트림이란?
- 자바 8부터 추가된 컬렉션의 저장 요소를 하나씩 참조하도록 도와주는 반복자입니다.
- 람다식으로 처리할 수 있도록 해주는 반복자입니다.
- 파일 입출력 stream과는 다른 개념입니다

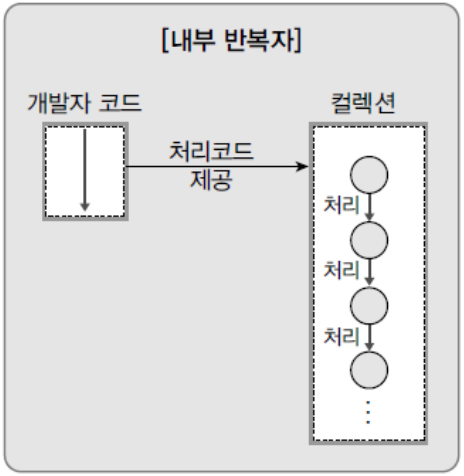
스트림의 특징
Iterator와 비슷한 역할을 하는 반복자 입니다.
대부분 메서드는 함수적 인터페이스 타입 입니다.
속도면에서 빠릅니다.



일반적 Iterator



컬렉션 Iterator



컬렉션에서 stream사용의 간단 예제

기존의 코드

```
List<String> list = new ArrayList<>();
list.add("홍길동");
list.add("박찬호");
list.add("이순신");
list.add("홍길자");

Iterator<String> iter = list.iterator(); //반복자
while(iter.hasNext()) {
      System.out.println(iter.next() );
}
```

Stream() 반복자 코드

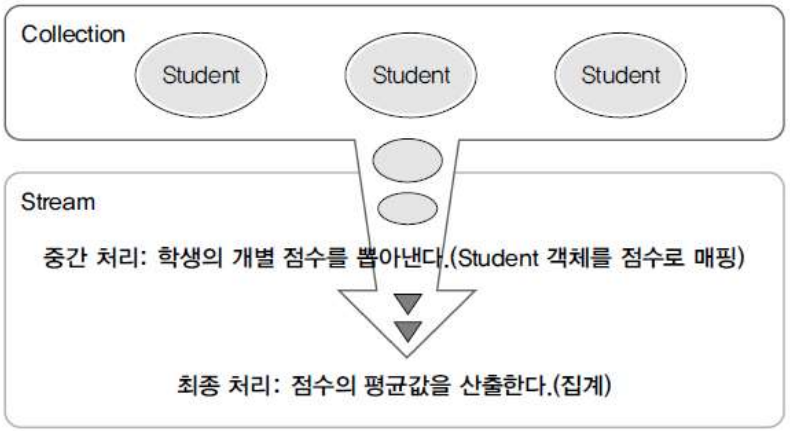
```
List<String> list = new ArrayList<>();
list.add("홍길동");
list.add("박찬호");
list.add("이순신");
list.add("홍길자");

Stream<String> stream = list.stream();
stream.forEach( (name) -> System.out.println(name));
```

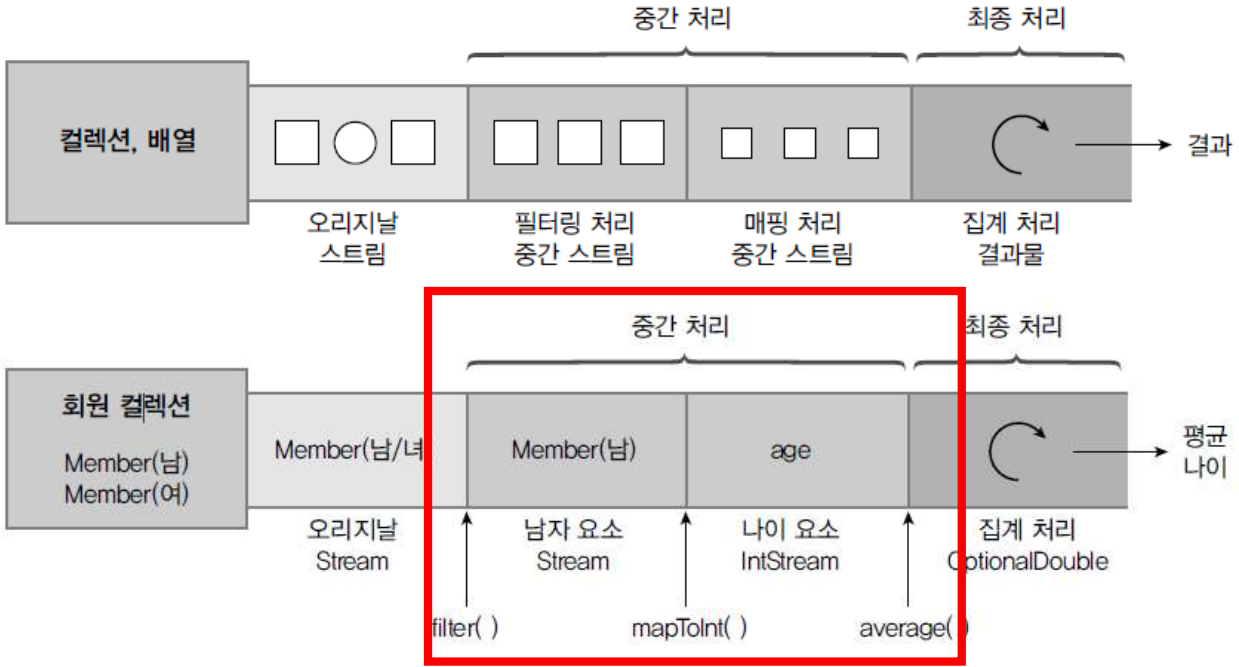
결과

홍길동
박찬호
이순신
홍길자

Stream은 중간처리와 최종처리가 함수형으로 표현가능



파이프라인 – 여러 개의 stream이 연결되어 있는 구조



중간처리 메서드

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

distinct( ) 메소드
– 중복을 제거하는 기능

filter( ) 메소드
-매개값으로 주어진 Predicate가 true를 리턴하는 요소만 필터링

mapXXX( ) 메소드
-요소를 대체하는 요소로 구성된 새로운 스트림 리턴

sorted( ) 메소드
-요소를 정렬

최종 처리메서드

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

forEach( ) 메서드
- 결과를 출력할 때 사용

Collect() 메서드
-컬렉션 객체를 List, Set, Map으로 변환처리 할 때 사용한다

Chapter 22

수고하셨습니다