# Chapter 8. 스트림

## 74장. 함수형 프로그래밍 방식과 람다식 문법

- 자바는 객체 지향 프로그래밍 : 기능을 수행하기 위해서는 객체를 만들고 그 객체 내부에 멤버 변수를 선언하고 기능을 수행하는 메서드를 구현
- 자바 8부터 함수형 프로그래밍 방식을 지원하고 이를 람다식(Lambda expression)이라 함
- 함수의 구현과 호출만으로 프로그래밍이 수행되는 방식
- 함수형 프로그래밍(Functional Programming: FP)

> 함수형 프로그래밍은 순수함수(pure function)를 구현하고 호출함으로써 외부 자료에 부수적인 영향(side effect)를 주지 않도록 구현하는 방식입니다. 순수 함수란 매개변수만을 사용하여 만드는 함수입니다. 즉, 함수 내부에서 함수 외부에 있는 변수를 사용하지 않아 함수가 수행되더라도 외부에는 영향을 주지 않습니다.
> 함수를 기반으로 하는 프로그래밍이고 입력 받는 자료 이외에 외부 자료를 사용하지 않아 여러 자료가 동시에 수행되는 병렬처리가 가능합니다. 함수형 프로그래밍은 함수의 기능이 자료에 독립적임을 보장합니다. 이는 동일한 자료에 대해 동일한 결과를 보장하고, 다양한 자료에 대해 같은 기능을 수행할 수 있습니다.

▶ 람다식(Lambda expression)

① 매개 변수가 하나인 경우 자료형과 괄호 생략 가능 : str-> {System.out.println(str); }
② 매개 변수가 두개 이상인 경우 중괄호를 생략할 수 없음 : x, y -> {System.out.println(x+y); } //오류
③ 실행문이 한 문장인 경우 중괄호 생략 가능 : str-> System.out.println(str);
④ 실행문이 한 문장이라도 return문이 있으면 중괄호 생략 불가 : str -> return str.length(); //오류
⑤ 실행문이 한 문장의 반환문인 경우 return 과 중괄호 모두 생략 가능 : str -> str.length();

▶ 람다식을 테스트하기 위해 인터페이스를 이용해 구현합니다.

```
public interface Calc {
    int calc(int x, int y);
}

public class LambdaExpressionTest {
    public static void main(String[] args) {
        Calc add = (x, y) -> {return x+y;};
        Calc minus = (x, y) -> x-y;
        Calc multply = (x, y) -> {return x*y;};

        System.out.println(add.calc(3,2));
        System.out.println(minus.calc(3, 2));
```

```
        System.out.println(multply.calc(3,2));
    }
}
```

▶ 함수형 인터페이스 선언하기

① 람다식을 선언하기 위한 인터페이스는 익명 함수와 매개 변수만으로 구현되므로 인터페이스는 단 하나의 메서드만을 선언해야함(모호함 발생/혼돈)

② @FunctionalInterface 애노테이션(annotation) : 선언 시 부터 오류 검출 가능

③ 함수형 인터페이스라는 의미, 내부에 여러 개의 메서드를 선언하면 에러남

▶ 두 수를 입력받아 큰 수와 작은수를 리턴하는 람다표현식 작성

```
@FunctionalInterface
public interface MaxorMinNumber {
    int maxOrMin(int x, int y);
}

public class MaxNumTest {
    public static void main(String[] args) {
        MaxorMinNumber maxNum = (x, y) -> {
            if(x > y){
                return x;
            } else {
                return  y;
            }
        };
        MaxorMinNumber minNum = (x, y) -> x < y? x : y;

        System.out.println("Max = " + maxNum.maxOrMin(20, 10));
        System.out.println("Min = " + minNum.maxOrMin(20, 10));
    }
}
```

# 75장. 스트림(Stream)

▶ 스트림이란?

① 자료의 대상과 관계없이 동일한 연산을 수행
- 배열, 컬렉션을 대상으로 연산을 수행 함
- 일관성 있는 연산으로 자료의 처리를 쉽고 간단하게 함
- 자료 처리에 대한 추상화가 구현되었다고 함

② 한번 생성하고 사용한 스트림은 재사용 할 수 없음
- 자료에 대한 스트림을 생성하여 연산을 수행하면 스트림은 소모됨
- 다른 연산을 수행하기 위해서는 스트림을 다시 생성해야 함

③ 스트림 연산은 기존 자료를 변경하지 않음
- 자료에 대한 스트림을 생성하면 스트림이 사용하는 메모리 공간은 별도로 생성되므로 연산이 수행되도 기존 자료에 대한 변경은 발생하지 않음

④ 스트림 연산은 중간 연산과 최종 연산으로 구분 됨
- 스트림에 대해 중간 연산은 여러 개의 연산이 적용될 수 있지만 최종 연산은 마지막에 한 번만 적용됨
- 최종연산이 호출되어야 중간 연산에 대한 수행이 이루어 지고 그 결과가 만들어짐
- 따라서 중간 연산에 대한 결과를 연산 중에 알수 없음
- 이를 '지연 연산'이라 함

▶ 정수 배열에 스트림을 생성하여 연산을 수행 하는 예

```
import java.util.Arrays;

public class IntArrayTest {
	public static void main(String[] args) {
		int[] arr = {1, 2, 3, 4, 5};
		int sumVal = Arrays.stream(arr).sum();
		long count = Arrays.stream(arr).count();
		System.out.println(sumVal);
		System.out.println(count);

		for(int i : arr){
			System.out.println(i);
		}

		System.out.println("==================");

		Arrays.stream(arr).forEach(i -> System.out.print(i + "\t"));
	}
}
```

▶ 중간연산과 최종연산

① 중간 연산과 최종 연산에 대한 구현은 람다식(→)을 활용함

② 중간 연산의 예 - filter(), map(), sorted() 등

- 조건에 맞는 요소를 추출(filter)하거나 요소를 변환 함(map)
- 최종 연산이 호출될 때 중간 연산이 수행되고 결과가 생성 됨

③ 최종 연산의 예 - forEach(), count(), sum() 등

- 스트림이 관리하는 자료를 하나씩 소모해가며 연산이 수행 됨
- 최종 연산 후에 스트림은 더 이상 다른 연산을 적용할 수 없음
- forEach() : 요소를 하나씩 꺼내 옴
- count() : 요소의 개수
- sum() : 요소들의 합

▶ ArrayList에 문자열 자료(이름)을 넣고 이에 대한 여러 연산을 수행해보기

```
package streamTest;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Stream;

public class ArrayListStreamTest {
	public static void main(String[] args) {
		List<String > list = new ArrayList<>();
		list.add("Tomas");
		list.add("Adward");
		list.add("Jack");

		// 리스트 내용 출력하기
		list.stream.forEach(s -> System.out.print(s + "₩t"));
		System.out.println();

		// 리스트 내용 정렬하여 출력하기
		list.stream().sorted().forEach(s-> System.out.print(s + "₩t"));
		System.out.println();

		// 리스트 내용의 각 길이 출력하기
		list.stream().map(k->k.length()).forEach(k -> System.out.printf(k + "₩t"));
		System.out.println();

		// 문자열 길이가 5이상인 자료만 출력하기
		list.stream().filter(s -> s.length() >= 5 ).sorted().forEach(s-> System.out.printf(s + "₩t"));
```

```
        System.out.println();
        // 정수자료에 대한 처리 예
        List<Integer> intList = new ArrayList<>();
        intList.add(1);
        intList.add(2);
        intList.add(3);
        intList.add(4);
        intList.add(5);

        // 전체 출력
        intList.stream().forEach(i -> System.out.println(i));
        System.out.println();

        // 합계출력(Integer(Object) -> primitive type 변환 필요)
        int sum = intList.stream().mapToInt(n->n.intValue()).sum();
        System.out.println("리스트의 합은 : " + sum);
    }
}
```

# 76장. 연산 수행에 대한 구현을 할 수 있는 reduce() 메서드

▶ reduce() : 정의된 연산이 아닌 프로그래머가 직접 구현한 연산을 적용

- reduce() 메서드의 두 번째 요소로 전달되는 람다식에 따라 다양한 기능을 수행 할 수 있음
- 람다식을 직접 구현하거나 람다식이 긴 경우 BinaryOperator를 구현한 클래스를 사용 함

▶ reduce() example

```
package streamTest;

import java.util.Arrays;
import java.util.function.BinaryOperator;

class CompareString implements BinaryOperator<String >{
    @Override
    public String apply(String s1, String s2) {
        if( s1.getBytes().length >= s2.getBytes().length) return s1;
        else return s2;
    }
}


public class ReduceTest {
    public static void main(String[] args) {
        String[] greetings = {"안녕하세요~~~", "hello", "Good morning", "반갑습니다^^"};
        String result;
        // 람다식을 이용한 reduce() 구현
        result = Arrays.stream(greetings).reduce("", (s1, s2)->
        {
            if( s1.getBytes().length >= s2.getBytes().length) return s1;
            else return s2;
        });
        System.out.println(result);

        System.out.println();

        // BinaryOperator를 구현하여 reduce() 수행하기
        result = Arrays.stream(greetings).reduce(new CompareString()).get();
        System.out.println(result);
    }
}
```

# 77장. 스트림을 활용하여 패키지 여행 비용 계산하기

▶ 문제

1. 여행사에 패키지 여행 상품이 있습니다.
   - 여행 비용은 15세 이상은 100만원, 그 미만은 50만원 입니다.
   - 고객 세 명이 패키지 여행을 떠난다고 했을 때 비용 계산과 고객 명단 검색 등에 대한 연산을 스트림을 활용하여 구현해 봅니다.
   - 고객에 대한 클래스를 만들고 ArrayList로 고객을 관리 합니다.

2. 고객 정보는 다음과 같습니다.

| | | | |
|---|---|---|---|
| CustomerLee | 이름 : 이순신 | 나이 : 40 | 비용 : 100 |
| CustomerKim | 이름 : 김유신 | 나이 : 20 | 비용 : 100 |
| CustomerHong | 이름 : 홍길동 | 나이 :13 | 비용 : 50 |

3. 출력 조건
   - 고객의 명단을 출력합니다.(전체 리스트 및 이름만...)
   - 여행의 총 비용을 계산합니다.
   - 고객 중 20세 이상인 사람의 이름을 정렬하여 출력합니다.

▶ 코드

```
public class TravelCustomerMain {
    public static void main(String[] args) {
        TravelCustomer customerLee = new TravelCustomer("이순신", 40, 100);
        TravelCustomer customerKim = new TravelCustomer("김유신", 20, 100);
        TravelCustomer customerHong = new TravelCustomer("홍길동", 13, 50);

        List<TravelCustomer> customerList = new ArrayList<>();

        customerList.add(customerLee);
        customerList.add(customerKim);
        customerList.add(customerHong);

        System.out.println("= 1. 고객 명단 추가된 순서대로 출력하기 ==");
        customerList.stream().forEach(x -> System.out.println(x));
        System.out.println("= 2. 고객 명단 추가된 순서대로 이름만 출력하기 ==");
        customerList.stream().map(x->x.getName()).forEach(x -> System.out.printf(x + " "));
        System.out.println();
        System.out.println("= 3. 총 여행 비용 출력하기 ==");
        int totalPrice = customerList.stream().mapToInt(x -> x.getPrice()).sum();
        System.out.println("총 여행경비는 = "  + totalPrice);
```

```
        System.out.println();
        System.out.println("= 4. 20세 이상 고객 명단 이름만 오름차순 출력 ==");
        customerList.stream().filter(x -> x.getAge()>=20)
                .map(x -> x.getName())
                .sorted()
                .forEach(x -> System.out.println(x));

        System.out.println("= 5. 20세 이상 고객 명단 전체 내림차순 정렬하여 출력 ==");
        customerList.stream().filter(x -> x.getAge()>=20)
                .sorted(Comparator.comparing(TravelCustomer::getName).reversed())
                .forEach(x -> System.out.println(x));
    }
}
```