14장. 람다식 & 스트림

Lambda & Stream

람다식(lambda expression)

함수형 프로그래밍과 람다식

- 자바는 객체를 기반으로 프로그램을 구현하며, 어떤 기능이 필요하다면 클래스를 만들고 그 안에 기능을 구현한 메서드(함수)를 만든 후 사용한다.
- 그러므로, 클래스가 없다면 메서드를 사용할 수 없다.
- **자바 8이후부터 사용할 수 있는 람다식은 객체 없이 인터페이스의 구현만으로 메서드를 호출할 수있다.**

람다식 구현하기

- 함수 이름이 없는 **익명 함수**를 만드는 것으로 익명 객체 구현.
- 표현식 : **(매개변수) -> { 실행문; }**

```
int add(int x, int y){
  return x + y;
}
```

일반 메서드(함수)

```
(x, y) ->  x + y
```

람다식

람다식(lambda expression)

람다식 문법 살펴보기

- **매개변수 자료형과 괄호 생략하기** : 매개변수가 하나인 경우 괄호 생략

```
(str) -> { System.out.println(str); }
```

- **중괄호 생략하기** : 중괄호 안의 구현 부분이 한 문장인 경우 중괄호 생략

```
str -> System.out.println(str);
```

- **매개변수가 없다면 괄호 생략할 수 없음**

```
( ) -> { 실행문; }
```

- **return 생략**

```
(x, y) -> {return x + y};

(x, y) -> x + y;
```

람다식(lambda expression)

함수형 인터페이스

람다식을 구현하기 위해 **함수형 인터페이스**를 만들고, 인터페이스에 람다식으로 구현할 메서드를 선언한다.

람다식은 이름이 없는 익명 함수로 구현하기 때문에 메서드에 **오직 하나의 추상 메서드**만 선언할 수 있다. (여러 개는 구현이 모호해지므로)

- **객체지향언어**는 객체를 기반으로 구현하는 방식
- **함수형 프로그램**은 함수를 기반으로 하고 자료를 입력받아 구현하는 방식

람다식(lambda expression)

- 객체 지향 방식으로 구현하기

```
package interface_impl;


public interface MyMath {

    public int myAbs(int n);
}
```

```
public class MyMathImpl implements MyMath{

    @Override
    public int myAbs(int n) {
        int value = (n < 0 ? -n : n); //절대값 연산
        return value;
    }

}
```

람다식(lambda expression)

- 객체 지향 방식으로 구현하기

```
public class MyMathTest {

    public static void main(String[] args) {

        MyMathImpl math = new MyMathImpl();
        System.out.println("절대값: " + math.myAbs(-4));
    }

}
```

람다식(lambda expression)

- 람다식으로 구현하기

```
package lambda;

@FunctionalInterface
public interface MyMath {
    //1개의 추상메서드만 사용 가능
    public int myAbs(int n);

    //public int mySquare(int n); //오류
}
```

애너테이션을 명시해서 실행전에 오류 체크

```
public class MyAbsTest {

    public static void main(String[] args) {
        //인터페이스형으로 객체 생성
        MyMath math;

        //절대값
        math = (x) -> (x < 0) ? -x : x;
        System.out.println("절대값: " + math.myAbs(-4));
    }
}
```

람다식(lambda expression)

- **객체 지향 프로그래밍 방식과 람다식 비교**

1. 객체 지향 프로그래밍 방식

```
package interface_impl;

public interface StringConcat {
    public void makeString(String s1, String s2);
}
```

```
public class StringConcatImpl implements StringConcat{

    @Override
    public void makeString(String s1, String s2) {
        System.out.println(s1 + ",  " + s2);
    }

}
```

람다식(lambda expression)

- **객체 지향 프로그래밍 방식과 람다식 비교**

1. 객체 지향 프로그래밍 방식

```
public class StringConctTest {

    public static void main(String[] args) {

        StringConcatImpl concat = new StringConcatImpl();
        concat.makeString("Hill", "State");
    }

}
```

람다식(lambda expression)

- **객체 지향 프로그래밍 방식과 람다식 비교**

  2 람다식 방식

```
package lambda.concat;

@FunctionalInterface
public interface StringConcat {
    public void makeString(String s1, String s2);
}
```

```
public class StringConcatTest {

    public static void main(String[] args) {
        String str1 = "Hill";
        String str2 = "State";

        StringConcat concat;
        concat = (s, v) -> System.out.println(s + ", " + v);
        concat.makeString(str1, str2);
    }
}
```

람다식(lambda expression)

함수형 인터페이스를 변수 및 반환 자료형으로 사용

```
@FunctionalInterface
interface PrintString{

    void showString(String str);
}

public class LambdaTest {

    public static void main(String[] args) {
        //인터페이스형 객체 선언 - 람다식 구현
        PrintString lambdaPrint;
        lambdaPrint = (str) -> System.out.println(str);
        lambdaPrint.showString("Good Luck!!");

        //void형 메서드 호출
        writeString(lambdaPrint);
```

람다식(lambda expression)

```
        //return이 있는 메서드 호출
        PrintString prStr = returnPrint();
        prStr.showString("Good Luck!!");
    }

    //함수형 인터페이스를 매개변수로 전달
    public static void writeString(PrintString prStr) {
        prStr.showString("Good Luck!!");
    }

    //함수형 인터페이스를 반환 자료형으로 사용
    public static PrintString returnPrint() {
        return str -> System.out.println(str);
    }
```

```
Good Luck!!
Good Luck!!
Good Luck!!
```

스트림(Stream)

- **스트림(Stream)**
  - 자료가 모여있는 배열이나 컬렉션에서 처리에 대한 기능을 구현해 놓은 클래스
  - 스트림 클래스는 람다식으로 처리하는 반복자이다.

Module java.base

Package java.util.stream

Classes to support functional-style operations on streams of elements, s

```
int sum = widgets.stream()
                 .filter(b -> b.getColor() == RED)
                 .mapToInt(b -> b.getWeight())
                 .sum();
```

asList

```
@SafeVarargs public static <T> List<T> asList(T... a)
```

Returns a fixed-size list backed by the specified array. (Changes t
combination with `Collection.toArray()`. The returned list is se

This method also provides a convenient way to create a fixed-size

```
List<String> stooges = Arrays.asList("Larry", "Moe
```

스트림(Stream)

- **스트림(Stream)**

```
public class StreamTest {

    public static void main(String[] args) {
        /*ArrayList<String> companyList = new ArrayList<>();
        companyList.add("LG");
        companyList.add("Samsung");
        companyList.add("Hyundai");*/

        //Arrays 클래스 사용
        List<String> companyList = Arrays.asList("LG", "Samsung", "현대");

        for(String company : companyList)
            System.out.println(company);

        //Stream 클래스 - 람다식으로 구현
        Stream<String> stream = companyList.stream();
        stream.forEach(company -> System.out.println(company));
    }
}
```

스트림(Stream)

- 스트림(Stream)

```
public class ArraysTest {

    public static void main(String[] args) {
        int[] num1 = {3, 1, 2, 4};

        System.out.println(Arrays.toString(num1));

        //num1의 요소 중 2개 복사
        int[] num2 = Arrays.copyOf(num1, 2);
        System.out.println(Arrays.toString(num2));

        //num1 오름차순 정렬하기
        Arrays.sort(num1);
        System.out.println(Arrays.toString(num1));
    }
}
```

스트림(Stream)

- 배열로 부터 스트림 얻기

```java
//문자형 배열
String[] fruit = {"apple", "banana", "grape"};

for(int i = 0; i < fruit.length; i++) {
    System.out.println(fruit[i]);
}

//stream의 forEach()와 람다식으로 구현
Stream<String> strStream = Arrays.stream(fruit);
strStream.forEach(str -> System.out.println(str));

//정수형 배열
int[] num = {1, 2, 3, 4};

for(int n : num)
    System.out.println(n);

//스트림을 얻어서 출력
Arrays.stream(num).forEach(n -> System.out.println(n));

//스트림을 얻어서 합계 구하기
int sumVal = Arrays.stream(num).sum();
int count = (int) Arrays.stream(num).count(); //count() long형 반환
double avg = (double)sumVal / count;

System.out.println("합계 : " + sumVal);
System.out.println("개수 : " + count);
System.out.println("평균 : " + avg);
```

스트림(Stream)

- **스트림(Stream) 연산**

  중간 연산 – 자료를 거르거나 변경하여 또 다른 자료를 내부적으로 생성함

  최종 연산 – 생성된 내부 자료를 소모해 가면서 연산을 수행함

중간연산 – filter(), map()

- 문자열 배열이 있을 때 문자열의 길이가 5 이상인 경우 출력

```
sList.stream().filter(s->s.length() >=5).forEach(s->System.out.println(s));
```

- 고객 클래스에서 고객 이름만 가져와 출력

```
customerList.stream().map(c->c.getName()).forEach(s->System.out.println(s));
```

최종 연산 – forEach(), count(), sum(), reduce()

- List컬렉션에서 정렬후 출력

```
sList.stream().sorted().forEach(str -> System.out.print(str+" "));
```

스트림(Stream)

- 컬렉션으로 스트림 얻기

```java
package stream.kind;

public 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;
    }
}
```

스트림(Stream)

- **컬렉션으로 스트림 얻기**

```
public class StudentStreamTest {

    public static void main(String[] args) {
        List<Student> list = Arrays.asList(
            new Student("콩쥐", 90),
            new Student("팥쥐", 70),
            new Student("심청", 80)
        );

        //Student로 부터 스트림 얻어 출력하기
        Stream<Student> stdStream = list.stream();
        stdStream.forEach(std -> {
            //System.out.println(std.getName() + " : " + std.getScore());
            String name = std.getName();
            int score = std.getScore();
            System.out.println(name + " : " + score);
        });
```

스트림(Stream)

- **컬렉션으로 스트림 얻기**

```
//스트림 연산
//stream()은 한 번 사용하면 소모되므로 다시 값을 할당함
System.out.println("=== 학생의 이름(매핑) 출력 ===");
stream = list.stream();
stream.map(std -> std.getName())
      .forEach(s -> System.out.println(s));

System.out.println("=== 학생의 점수(매핑) 출력 ===");
stream = list.stream();
stream.mapToInt(std -> std.getScore())
      .forEach(s -> System.out.println(s));

System.out.println("=== 점수가 90 이상인 학생 이름(필터링) ===");
list.stream().filter(std -> std.getScore() >= 90)
             .map(std -> std.getName())
             .forEach(s -> System.out.println(s));
```

```
콩쥐 : 90
팥쥐 : 70
심청 : 80
=== 학생의 이름 출력 ===
콩쥐
팥쥐
심청
=== 학생의 점수 출력 ===
90
70
80
=== 점수가 90 이상인 학생 이름 필터링 ===
콩쥐
```

스트림(Stream)

- **스트림을 활용하여 여행객의 여행 비용 계산하기**

CustomerLee	CustomerKang	CustomerHong
- 이름 : 이순신 - 나이 : 40 - 비용 : 100만원	- 이름 : 강감찬 - 나이 : 30 - 비용 : 100만원	- 이름 : 홍길동 - 나이 : 14 - 비용 : 50만원

▶ 예제 시나리오

1. 고객의 명단을 출력합니다.
2. 여행의 총 비용을 계산합니다.
3. 고객 중 20세 이상인 사람의 이름을 정렬하여 출력합니다.

스트림(Stream)

- 스트림을 활용하여 여행객의 여행 비용 계산하기

```
public class Customer {
    private String name;
    private int age;
    private int price;

    public Customer(String name, int age, int price) {
        this.name = name;
        this.age = age;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getPrice() {
        return price;
    }
}
```

스트림(Stream)

- **스트림을 활용하여 여행객의 여행 비용 계산하기**

```
List<Customer> customerList = new ArrayList<>();

Customer lee = new Customer("이순신", 40, 100);
Customer kang = new Customer("강감찬", 10, 100);
Customer hong = new Customer("홍길동", 15, 50);

customerList.add(lee);
customerList.add(kang);
customerList.add(hong);

System.out.println("===고객명단 추가된 순서대로 출력 ===");
customerList.stream().map(c -> c.getName()).forEach(s -> System.out.println(s));

int total = customerList.stream().mapToInt(c->c.getPrice()).sum();
System.out.println("총 여행 비용은 : " + total + "입니다.");

System.out.println("===20세 이상 고객 명단 정렬하여 출력===");
customerList.stream().filter(c->c.getAge() >= 20).map(c->c.getName())
            .sorted().forEach(s->System.out.println(s));
```

```
===고객명단 추가된 순서대로 출력 ===
이순신
강감찬
홍길동
총 여행 비용은 : 250입니다.
===20세 이상 고객 명단 정렬하여 출력===
이순신
```