Java의 신 문법

Kim Hye Kyung
topickim@naver.com

Enum

enum - 고정 데이터 사용을 위한 개발 방법

상수값으로 선언

public static final 상수로 선언

```
class Color{
        public static final int RED = 1;
        public static final int YELLOW=2;
}
```

열거형 클래스 사용

enum class명{ ... }로 구현

```
enum Color{
        RED, YELLOW
}
```

enum 클래스

- 열거형 클래스 구현 방법
  - class 키워드 대신 enum 키워드 사용
  - 열거형 상수 선언(enumeration constants)
    - public static final 자동 반영
    - ,(콤마)로 구분

```
//enum 클래스
enum Color{
        //열거 상수
        RED, YELLOW
}
```

- 예시

```
3  public enum Color {
4      RED,YELLOW,BLUE
5  }
6
7
8
9
```

```
3  public class EnumUseTest {
4
5      public static void main(String[] args) {
6          System.out.println(Color.RED); //RED
7
8          //열거 상수 목록을 Color[] 형태로 반환
9          //RED   YELLOW  BLUE 순으로 출력
10         Color[] colors = Color.values();
11         for(Color v : colors) {
12             System.out.println(v);
13         }
14     }
15
16 }
```

람다

람다식

- JDK1.8 부터 추가된 표현식
- 식별자 없이 실행 가능한 이름 없는 함수 [익명 함수]
- 필요성
  - 구현 코드의 간소화
  - 가독성 유지보수 향상
- 기본 Syntax

```
( parameters ) -> expression body
( parameters ) -> { expression body }
() -> { expression body }
() -> expression body
...
```

람다식 구현

- 일반 interface 구현과 람다식 구현은 동일한 기능 수행

```
3  interface Calculation {
4      //4칙 연산 수행후 반환하는 추상 메소드 선언
5      int operation(int v1, int v2);
6  }
7  interface MessageService {
8      void sayMessage(String message);
9  }
```

```
private static int operate(int v1, int v2, Calculation calc) {
    return calc.operation(v1, v2);
}

public static void main(String args[]) {
    // 타입 선언한 람다식
    Calculation addition = (int v1, int v2) -> v1 + v2;
    System.out.println("5 + 5 = " + operate(5, 5, addition));//5 + 5 = 15

    // 타입 선언없는 람다식
    Calculation subtraction = (v1, v2) -> v1 - v2;
    System.out.println("5 - 5 = " + operate(5, 5, subtraction));//5 - 5 = 0

    // 중괄호와 리턴문 있는 람다식
    Calculation multiplication = (int v1, int v2) -> {
        return v1 * v2;
    };
    System.out.println("5 x 5 = " + operate(5, 5, multiplication));//5 x 5 = 25

    // 중괄호와 리턴문 없는 람다식
    Calculation division = (int v1, int v2) -> v1 / v2;
    System.out.println("5 / 5 = " + operate(5, 5, division));//5 / 5 = 1
```

생각해 보기?

정통 코드로 구현시 어떤 형태의 문법으로 구현해야 하는가?

람다식 코드로 원리 이해하기

- 일반 interface 구현과 람다식 구현은 동일한 기능 수행

```
3  interface Calculation {
4      //4칙 연산 수행후 반환하는 추상 메소드 선언
5      int operation(int v1, int v2);
6  }
7  interface MessageService {
8      void sayMessage(String message);
9  }
```

```
private static int operate(int v1, int v2, Calculation calc) {
    return calc.operation(v1, v2);
}

public static void main(String args[]) {
    // 타입 선언한 람다식
    Calculation addition = (int v1, int v2) -> v1 + v2;
    System.out.println("5 + 5 = " + operate(5, 5, addition));//5 + 5 = 15

    // 중괄호와 리턴문 없는 람다식
    Calculation division = (int v1, int v2) -> v1 / v2;
    System.out.println("5 / 5 = " + operate(5, 5, division));//5 / 5 = 1
```

Calculation인터페이스의하위클래스 implements Calculation{

public int operation(int v1, int v2){

return v1 + v2;

}

}

= 25

람다식 코드로 원리 이해하기

- 일반 interface 구현과 람다식 구현은 동일한 기능 수행

```
3  interface Calculation {
4      //4칙 연산 수행후 반환하는 추상 메소드 선언
5      int operation(int v1, int v2);
6  }
7  interface MessageService {
8      void sayMessage(String message);
9  }
```

```
private static int operate(int v1, int v2, Calculation calc) {
    return calc.operation(v1, v2);
}

public static void main(String args[]) {
    // 타입 선언한 람다식
    Calculation addition = (int v1, int v2) -> v1 + v2;
    System.out.println("5 + 5 = " + operate(5, 5, addition));//5 + 5 = 15

    // 타입 선언없는 람다식
    Calculation subtraction = (v1, v2) -> v1 - v2;
    System.out.println("5 - 5 = " + operate(5, 5, subtraction));//5 - 5 = 0
```

```
Calculation인터페이스의하위클래스 implements Calculation{
        public int operation(int v1, int v2){
            return v1   - v2;
        }
}
```

25

생각해 보기?

addition 변수와 subtraction 변수값 출력시 출력되는 데이터는?

람다식 코드로 원리 이해하기

생각해 보기?

addition 변수와 subtraction 변수값 출력시 출력되는 데이터는?

```
private static int operate(int v1, int v2, Calculation calc) {
    return calc.operation(v1, v2);
}

public static void main(String args[]) {
    // 타입 선언한 람다식
    Calculation addition = (int v1, int v2) -> v1 + v2;
    System.out.println("5 + 5 = " + operate(5, 5, addition));//5 + 5 = 15

    // 타입 선언없는 람다식
    Calculation subtraction = (v1, v2) -> v1 - v2;
    System.out.println("5 - 5 = " + operate(5, 5, subtraction));//5 - 5 = 0
```

```
// 람다식으로 생성된 객체의 주소값
System.out.println(addition);//syntax.study.RamdaSyntax$$Lambda$1/303563356@3e3abc88
System.out.println(subtraction);//syntax.study.RamdaSyntax$$Lambda$2/1826771953@53d8d10a
```

Stream API

Stream API란?

- java.uitl.stream package
- 배열이나 컬렉션처럼 데이터 그룹을 간단하고 효율적으로 처리할 수 있도록 JDK 8부터 지원
- 스트림 데이터의 특징
  - 처리 과정에서 임시로 존재
  - 작업 후 자동 소멸
  - 데이터 소스 원본의 변경없이 데이터 처리 작업 수행
  - 지연 연산 가능

Stream API 사용을 위한 주요 클래스 : Optional

- java.util.Optional
- null을 대신하기 위해 만들어진 새로운 코어 라이브러리 데이터 타입
  - null이나 null이 아닌 값을 담을 수 있는 클래스
  - 객체를 보유하고 있는 container 기능
    - 보유한 객체 활용 메소드 제공
- Optional 클래스에서는 3가지 정적 팩토리 메서드를 제공
  - Optional.empty() : 빈 Optional 객체 생성
  - Optional.of(value) : value값이 null이 아닌 경우에 사용
  - Optional.ofNullable(value) : value값이 null인지 아닌지 확실하지 않은 경우에 사용

Optional 클래스

- Optional 객체 생성
  - Optional.empty()
    - 빈 Optional 객체 생성
  - Optional.of(value)
    - value값이 null이 아닌 경우에 사용
    - null인 경우 NullPointerException 발생
  - Optional.ofNullable(value)
    - value값이 null인지 아닌지 확실하지 않은 경우에 사용
    - null이 넘어 올 경우에는 empty Optional 객체를 생성

```java
//empty() : 빈 Optional 객체 생성
Optional<String> opt1 = Optional.empty();
System.out.println(opt1); //Optional.empty

//of() : value값이 null이 아닌 경우에 사용, null인 경우 NullPointerException 발생
Optional<String> opt2 = Optional.of("문자열 데이터");
System.out.println(opt2);   //Optional[문자열 데이터]


Optional<String> opt3NotNull = Optional.of(null); //NullPointerException 발생
System.out.println(opt3NotNull);

//ofNullable() : null인 경우 empty Optional 객체 생성
Optional<String> opt4Null = Optional.ofNullable(null);
System.out.println(opt4Null);   //Optional.empty

opt4Null = Optional.ofNullable("문자열 데이터");
System.out.println(opt4Null);   //Optional[문자열 데이터]
```

Optional 클래스

- Optional이 보유한 객체 활용 API

메소드명	Optional 객체가 보유한 데이터	특징
ifPresent	객체를 보유한 경우	객체값 반환
	null	실행 생략
orElse	객체를 보유한 경우	객체값 반환
	null	orElse 메소드의 parameter값 반환
orElseThrow	객체를 보유한 경우	객체값 반환
	null	orElseThrow 메소드의 paramete로 선언된 RuntimeException 예외 발생

```
Optional<String> v1 = Optional.ofNullable("문자열");
v1.ifPresent(v -> System.out.println(v.charAt(0))); //문

Optional<String> v2 = Optional.ofNullable(null);
v2.ifPresent(v -> System.out.println(v.charAt(0))); //null인 경우 실행 안 함
```

```
Optional<String> v3 = Optional.ofNullable(null);
System.out.println(v3.orElse("null인 경우에만 반환")); //null인 경우에만 반환

Optional<String> v4 = Optional.ofNullable("문자열 데이터");
System.out.println(v4.orElse("null인 경우에만 반환")); //문자열 데이터
```

```
Optional<String> v5 = Optional.ofNullable("문자열 데이터");
System.out.println(v5.orElseThrow(IllegalArgumentException::new)); //문자열 데이터

Optional<String> v6 = Optional.ofNullable(null);
System.out.println(v6.orElseThrow(IllegalArgumentException::new)); //IllegalArgumentException 발생
```

Stream API 사용 3단계

Stream 생성

Stream 중개 연산 : 파이프필터 패턴 적용

Stream 최종 연산

```
int [] values = {1, 2, 3, 4, 5, 6};
Arrays.stream(values)        //Stream 생성
.filter(v -> v % 2 == 0 )    //중개 연산
.sum();                      //최종 연산
```

Stream 생성 방법

- 데이터 소스의 종류에 따라 스트림을 생성하는 메소드가 다름

배열	컬렉션
java.util.Arrays의 stream() 메소드 사용 **Stream<배열요소의타입> 변수 = Arrays.stream(배열);**	java.util.Collection의 stream() 메소드 사용 **Stream<컬렉션요소의타입> 변수 = 컬렉션.stream();**

Stream의 주요 API

- 객체 타입의 데이터를 처리하기 위한 Stream API

메소드	특징
long count()	스트림 요소들의 개수를 반환
Stream<T> filter(Predicate<? super T> predicate)	Stream 요소들을 기반으로 조거넹 만족하는 요소만 선택해서 새로운 Stream 생성 ist객체.stream().filter(i -> 남기고자 하는 조건으로 true 반환)
void forEach(Consumer<? super T> action	Stream의 모든 요소에 대한 반복문을 실행
Stream map(Function mapper)	Stream의 요소들을 조건에 지정된 값으로 변환하여 새로운 Stream 생성 list객체.stream().map(i -> 값을 변화시킬 패턴 또는 함수)
Optional<T> reduce(BinaryOperator<T> accumulator)	Stream 요소들을 기반으로 조건에 따라 처리된 결과값을 Optional 객체로 반환
Stream<T> sorted()	Stream의 요소들을 정렬한 후 새로운 Stream으로 반환
Object[] toArray()	Stream의 요소들을 배열로 변환 후 반환
Optional<T> reduce()	list객체.stream().reduce((i1, i2) -> 연속된 2개의 아이템을 어떻게 1개의 아이템으로 합병할 것인지의 함수)

Stream의 주요 API

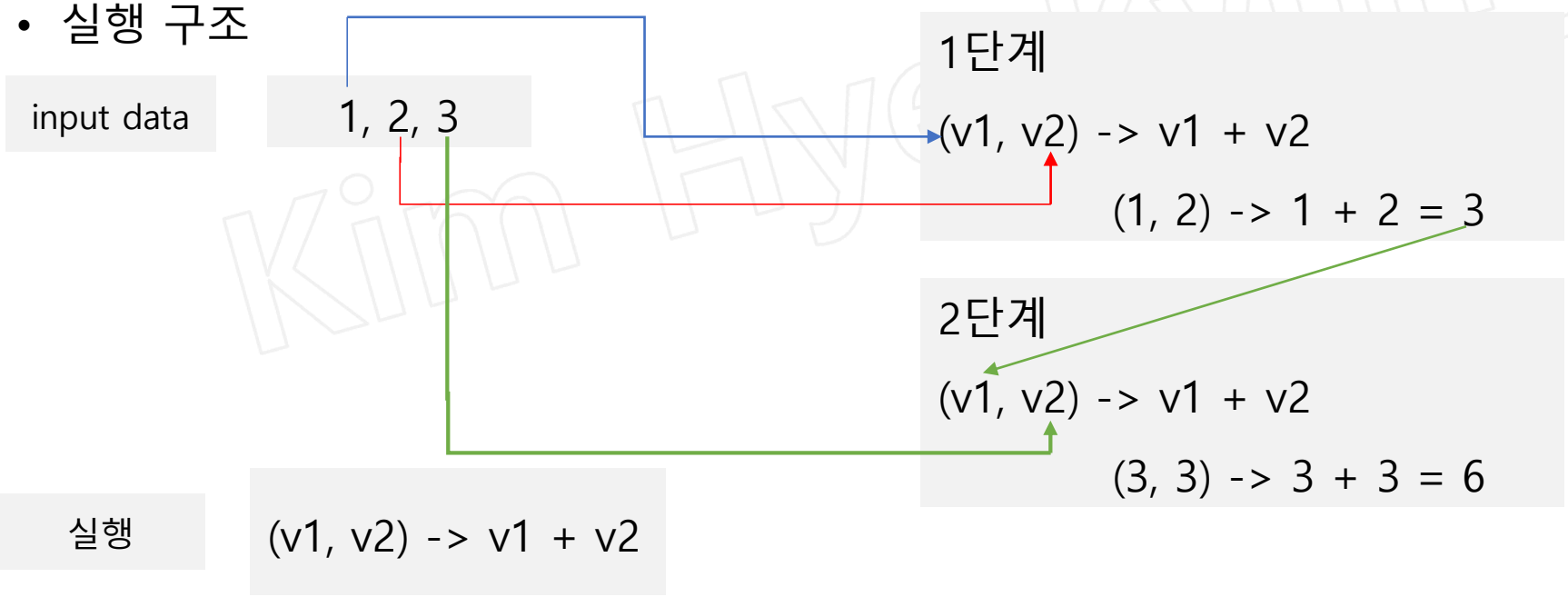
- 예제로 API 이해하기

```
System.out.println("--- forEach : 반복 ---");
Arrays.asList(1, 2, 3).stream().forEach(System.out::println); // 1 2 3

System.out.println("--- map : 리스트에 있는 요소의 제곱 연산 및 출력 ---");
Arrays.asList(1, 2, 3).stream().map(i -> i*i).forEach(System.out::println);  // 1 4 9

System.out.println("--- skip : 인덱스까지의 요소를 제외하고 새로운 Stream 생성 및 출력 ---");
Arrays.asList(1, 2, 3).stream().skip(2).forEach(System.out::println); //3

System.out.println("--- limit : 선언한 인덱스까지의 요소 추출 및 출력 ---");
Arrays.asList(1, 2, 3).stream().limit(2).forEach(System.out::println); //1 2

System.out.println("--- filter : Stream의 요소마다 조건식을 만족하는 요소로만 구성된 Stream 반환 및 출력 ---");
Arrays.asList(1, 2, 3).stream().filter(i -> i <= 2).forEach(System.out::println);// 1 2

System.out.println("--- flatMap : Stream 내부에 있는 객체들을 연결하여 Stream 반환 ---");
Arrays.asList(Arrays.asList(1, 2), Arrays.asList(10, 20), Arrays.asList(100, 200)).stream()
                .flatMap(i -> i.stream()).forEach(System.out::println);// 1 2 10 20 100 200

System.out.println("--- reduce : Stream의 단일 요소로 반환 ---");
System.out.println(Arrays.asList(1, 2, 3).stream().reduce((v1, v2) -> v1 + v2).get()); //6
System.out.println(Arrays.asList(1, 2, 3).stream().reduce((v1, v2) -> v2 - v1).get()); //2

System.out.println("--- collect() or iterator() : 콜렉션 객체를 만들어 반환 ---");
System.out.println(Arrays.asList(1,2,3).stream().collect(Collectors.toList())); //[1, 2, 3]
Arrays.asList(1,2,3).stream().iterator().forEachRemaining(System.out::println); // 1 2 3
```

```
--- forEach : 반복 ---
1
2
3
--- map : 리스트에 있는 요소의 제곱 연산 및 출력 ---
1
4
9
--- skip : 인덱스까지의 요소를 제외하고 새로운 Stream 생성 및 출력 ---
3
--- limit : 선언한 인덱스까지의 요소 추출 및 출력 ---
1
2
--- filter : Stream의 요소마다 조건식을 만족하는 요소로만 구성된 Stream 반환 및 출력 ---
1
2
--- flatMap : Stream 내부에 있는 객체들을 연결하여 Stream 반환 ---
1
2
10
20
100
200
--- reduce : Stream의 단일 요소로 반환 ---
6
2
--- collect() or iterator() : 콜렉션 객체를 만들어 반환 ---
[1, 2, 3]
1
2
3
```

Stream의 주요 API

- API 이해하기
  - 예제
    - Arrays.asList(1, 2, 3).stream().reduce((v1, v2) -> v1 + v2).get()
  - 실행 구조



Stream의 주요 API

- 예제로 API 이해하기

```
ArrayList<Integer> totalScore = new ArrayList<>();
totalScore.add(50);
totalScore.add(10);
totalScore.add(80);
totalScore.add(70);
totalScore.add(90);
totalScore.add(60);
totalScore.add(20);
```

```
System.out.print("1. 점수 목록 : ");
Stream<Integer> scores = totalScore.stream();
scores.forEach((n) -> System.out.print(n + " "));       // 1. 점수 목록 : 50 10 80 70 90 60 20

Optional<Integer> minScore = totalScore.stream().min(Integer::compare);
System.out.println("\n2. 최저 점수 : " + minScore.get());  // 2. 최저 점수 : 10

Optional<Integer> maxScore = totalScore.stream().max(Integer::compare);
System.out.println("3. 최고 점수 : " + maxScore.get());    // 3. 최고 점수 : 90

System.out.print("4. 점수 정렬 : ");
Stream<Integer> scores2 = totalScore.stream().sorted();
scores2.forEach((n) -> System.out.print(n + " "));      // 4. 점수 정렬 : 10 20 50 60 70 80 90

System.out.print("\n5. 낙제 점수 : ");
Stream<Integer> failScore = totalScore.stream().filter((n) -> n < 60);
failScore.forEach((n) -> System.out.print(n + " "));    // 5. 낙제 점수 : 50 10 20

System.out.print("\n6. 점수 합계 : ");
Optional<Integer> totalScoreSum = totalScore.stream().reduce((a, b) -> a + b);
System.out.println(totalScoreSum.get());                // 6. 점수 합계 : 380

System.out.print("7. 5점 추가 : ");
Stream<Integer> addScore = totalScore.stream().map((n) -> n + 5);
addScore.forEach((n) -> System.out.print(n + " "));     // 7. 5점 추가 : 55 15 85 75 95 65 25
System.out.println();

long cnt = totalScore.stream().count();
System.out.println("8. 점수 개수 : " + cnt);              // 8. 점수 개수 : 7

System.out.println("9. 점수 평균 : " + totalScoreSum.get() / cnt); // 9. 점수 평균 : 54
```

:: [더블 콜론]

Double colon Operation(::)

- Java 8에 추가된 메소드 연산자
- 메소드를 호출할 경우 타겟의 참조(객체)는 ::[double colon] 구분 기호 앞, 메소드 이름은 콜론 선언

참조(객체) :: 메소드명

```
People p1 = new People("박나래", 30);
People p2 = new People("유재석", 40);
People p3 = new People("현주엽", 50);
Arrays.asList(p1, p2, p3).forEach(System.out::println);
```

```
People(name=박나래, age=30)
People(name=유재석, age=40)
People(name=현주엽, age=50)
```

System.out 타입의 PrintStream의 println() 메소드의 prarmeter로 List내에 저장된 데이터를 하나씩 넘겨 주면서 출력하는 로직

Double colon Operation(::)

- 예제로 Double Colon Operation 이해하기

```java
People p1 = new People("박나래", 30);
People p2 = new People("유재석", 40);
People p3 = new People("현주엽", 50);

List<People> peoples = Arrays.asList(p1, p2, p3);

//step01
System.out.println("--- step01 : 기본 for문 ---");
for(People p : peoples) {
    System.out.println(p);
}

System.out.println("\n--- step02 : random식 ---");
peoples.forEach(p -> System.out.println(p));

System.out.println("\n--- step03 : double colon ---");
peoples.forEach(System.out::println);

System.out.println("\n--- step04 : double colon ---");
Arrays.asList(p1, p2, p3).forEach(System.out::println);
```

```
--- step01 : 기본 for문 ---
People(name=박나래, age=30)
People(name=유재석, age=40)
People(name=현주엽, age=50)

--- step02 : random식 ---
People(name=박나래, age=30)
People(name=유재석, age=40)
People(name=현주엽, age=50)

--- step03 : double colon ---
People(name=박나래, age=30)
People(name=유재석, age=40)
People(name=현주엽, age=50)

--- step04 : double colon ---
People(name=박나래, age=30)
People(name=유재석, age=40)
People(name=현주엽, age=50)
```