Enum, 제네릭, 람다식

Enum

정의 : Enum(Enumeration)은 열거라는 의미를 갖는다. 관련이 있는 상수들의 집합이다. 자바에서는 final로 String과 같은 문자열이나 숫자들을 나타내는 기본 자료형의 값을 고정할 수 있다. 이렇게 고정된 값을 **상수**라고 한다. 영어로는 constant입니다. 어떤 클래스가 상수만으로 작성되어 있으면 반드시 class로 선언할 필요는 없습니다. 이럴 때 class로 선언된 부분에 **enum**이라고 선언하면 이 객체는 상수의 집합이다. 라는 것을 명시적으로 나타낸다.

특징

1. 클래스를 상수처럼 사용할 수 있다.

```
public enum  Rank {
    THREE(3, 4_000),
    FOUR(4, 10_000),
    FIVE(5, 30_000);

    private final int match;
    private final int money;
    private int count;

    Rank(int match, int money) {  // Default 생성자는 private 으로 설정되어 있음.
        this.match = match;
        this.money = money;
    }
```

2. Enum 클래스를 구현하는 경우 상수 값과 같이 유일하게 하나의 인스턴스가 생성되어 사용된다.

```
public enum Rank {
    THREE(3, 4_000),
    FOUR(4, 10_000),
    FIVE(5, 30_000);

    private final int match;
    private final int money;
    private int count;

    Rank(int match, int money) {
        this.match = match;
        this.money = money;
    }

    public void plusCount() {
        this.count++;
    }
}
```

3. 서로 관련 있는 상수 값들을 모아 enum으로 구현하는 경우 유용하다.

4. 클래스와 같은 문법 체계를 따른다.
5. 상속을 지원하지 않는다.

Enum의 내부 Api

위에서 언급한 java.lang.Enum 클래스를 기본적으로 상속받고 있기 때문에 아래의 세 가지 메소드를 지원한다. (부모 클래스의 메소드라 사용 가능하다.)

1. values()

values() 는 Enum 클래스가 가지고 있는 모든 상수 값을 배열의 형태로 리턴 한다. 참고로 단순히 String 의 형태로 단순 반환하는 것이 아니라 인스턴스를 반환하는 것이다. 즉 Enum 클래스가 가지고 있는 모든 인스턴스를 배열에 담아 반환하는 것이다.

```
        public static void main(String[] args) {
                Rank[] values = Rank.values();
                for(int i = 0; i< values.length; i++) {
                        System.out.println(values[i]);
                }
    }

// 실행 결과 : THREE, FOUR, FIVE
```

2. valueOf()

valueOf() 메서드는 String 을 파라미터로 받는데 인자로 들어온 String 과 일치하는 상수 인스턴스가 존재하면 그 인스턴스를 반환한다. 이 또한 마찬가지로 단순히 문자열을 반환하는 것이 아니라 인자로 들어온 문자열과 일치하는 인스턴스를 반환하는 것이다.

```
        public static void main(String[] args) {
                System.out.println(Rank.valueOf("THREE"));
        }

// 실행 결과 : THREE
```

3. ordinal()

Enum 클래스 내부에 있는 상수들의 Index 를 리턴하는 메소드이다. 배열과 마찬가지로 0부터 인덱스가 시작하며 인덱스의 length 는 상수의 수 - 1 이다.

비교해서 보기

1. 그냥 상수

```
public static final int APPLE_FUJI = 0;
public static final int APPLE_PIPPIN = 1;
public static final int APPLE_GRANNY_SMITH = 2;

public static final int ORANGE_NAVEL = 0;
public static final int ORANGE_TEMPLE = 1;
public static final int ORANGE_BLOOD = 2;
```

2. enum

```
public enum Apple {FUJI, PIPPIN, GRANNY_SMITH, ORIGINAL}
public enum Orange {NAVEL, TEMPLE, BLOOD, ORIGINAL}
```

제네릭

정의 : 제네릭이란 JDK 1.5부터 도입한 클래스 내부에서 사용할 데이터 타입을 외부에서 지정하는 기법이다.

```
public static void main(String[] args) {
    List<String> words = new ArrayList<>();
}
```

왜 사용할까? ==========

```
public static void main(String[] args) {
    List numbers = Arrays.asList("1", "2", "3", "4", "5", "6");
    int sum = 0;
    for (Object number : numbers) {
        sum += (int) number;
    }
}
```

이런식으로 타입 없이 지정을 하면, 위 예제와 같이 List에 문자열을 넣어주어도 컴파일 에러가 발생하지 않고 런타임에 ClassCastException 이 터진다. 컴파일 시 타입을 체크하고 에러를 찾아낼 수 있는 컴파일 언어의 장점을 발휘하지 못하는 셈이다.

```
public static void main(String[] args) {
    List<Integer> numbers = Arrays.asList("1", "2", "3", "4", "5", "6");
    int sum = 0;
    for (Integer number : numbers) {
        sum += number;
    }
    System.out.println(sum);
}
```

컴파일 에러가 나타나서 디버깅에 용이해진다!

장점

1. 제네릭을 사용하면 잘못된 타입이 들어올 수 있는 것을 컴파일 단계에서 방지할 수 있다.

2. 클래스 외부에서 타입을 지정해주기 때문에 따로 타입을 체크하고 변환해줄 필요가 없다. 즉, 관리하기가 편하다.

3. 비슷한 기능을 지원하는 경우 코드의 재사용성이 높아진다.

보통 사용하는 암묵적 규칙 (자유로움)

타입	설명
<T>	Type
<E>	Element
<K>	Key
<V>	Value
<N>	Number

람다식

정의: 람다 함수는 프로그래밍 언어에서 사용되는 개념으로 **익명 함수**(Anonymous functions)를 지칭하는 용어이다. 현재 사용되고 있는 람다의 근간은 수학과 기초 컴퓨터과학 분야에서의 람다 대수이다. 람다 대수는 간단히 말하자면 수학에서 사용하는 함수를 보다 단순하게 표현하는 방법이다.

익명함수?

익명함수란 말그대로 함수의 이름이 없는 함수입니다. 익명함수들은 공통으로 일급객체(First Class citizen)라는 특징을 가지고 있습니다.

이 일급 객체란 일반적으로 다른 객체들에 적용 가능한 연산을 모두 지원하는 개체를 가르킵니다. 함수를 값으로 사용 할 수도 있으며 파라메터로 전달 및 변수에 대입 하기와 같은 연산들이 가능합니다.

장점 :

1. 코드의 간결성 - 람다를 사용하면 불필요한 반복문의 삭제가 가능하며 복잡한 식을 단순하게 표현할 수 있습니다.
2. 지연연산 수행 - 람다는 지연연상을 수행 함으로써 불필요한 연산을 최소화 할 수 있습니다.
3. 병렬처리 가능 - 멀티쓰레디를 활용하여 병렬처리를 사용 할 수 있습니다.

단점 :

1. 람다식의 호출이 까다롭습니다
2. 람다 stream 사용 시 단순 for문 혹은 while문 사용 시 성능이 떨어집니다.
3. 불필요하게 너무 사용하게 되면 오히려 가독성을 떨어 뜨릴 수 있습니다.

예제 :

기존 자바 문법

```
new Thread(new Runnable() {
   @Override
   public void run() {
      System.out.println("Welcome Heejin blog");
   }
}).start();
```

람다식 문법

```
new Thread(()->{
      System.out.println("Welcome Heejin blog");
}).start();
```