ENUM

열거형 데이터 타입 (enum)

- 몇 가지 한정된 **상수값으로 구성** 될 때 주로 사용
  - 예) 계절(봄, 여름, 가을, 겨울), 요일(월,화,수,목,금,토,일)
- enum은 내부적으로 **java.lang.Enum 클래스를 상속** 받고있다!
- 따라서, **다른 클래스를 상속 받을 수 없다.**
- 이러한 상수들을 'enum상수' 라고 불리며 **대문자로 작성**
- enum상수 하나하나가 **enum타입의 객체**가 된다.

enum의 사용과 특징

- 접근 제어자 public, default만 사용 가능
- 다른 클래스를 상속 받을 수 없다 (이미 java.lang.Enum상속)
- 여러 인터페이스를 구현하는 것은 가능
- 클래스 외부 뿐만 아니라 클래스 내부에서도 선언 가능

```
enum Week {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
```

enum의 필요성 (추가된 배경)

- **주석으로 상수**의 의미를 전달하고 있음
- 주석이 사라지거나 분리된다면

  숫자들이 어떤 것을 의미하는지 이해 X
- **이름만으로 의미를 전달** 할 수 있다면?

```java
public class Main {

    public static void main(String[] args) {
        /*
         * 월요일 == 1
         * 화요일 == 2
         * 수요일 == 3
         * 목요일 == 4
         * 금요일 == 5
         * 토요일 == 5
         * 일요일 == 6
         * */
        int day = 1;
        switch (day) {
        case 1:
            System.out.println("월요일 입니다.");
            break;
        case 2:
            System.out.println("화요일 입니다.");
            break;
            /*
             * ...생략...
             */
        }

    }
}
```

enum의 필요성 (추가된 배경)

- final을 사용 (바뀌지 않도록)
- static을 사용하여 **메모리에 한 번만 할당**

❖ **문제점**

- 월(month)에 대한 상수가 추가된다면?
- 같은 이름으로 정의된 상수가 있다면?

```java
public class Main {

    private final static int MONDAY     = 1;
    private final static int TUESDAY    = 2;
    private final static int WEDNESDAY  = 3;
    private final static int THURSDAY   = 4;
    private final static int FRIDAY     = 5;
    private final static int SATURDAY   = 6;
    private final static int SUBDAY     = 7;

    public static void main(String[] args) {
        int day = 1;
        switch (day) {
        case 1:
            System.out.println("월요일 입니다.");
            break;
        case 2:
            System.out.println("화요일 입니다.");
            break;
            /*
             * ...생략...
             */
        }

    }
}
```

enum의 필요성 (추가된 배경)

- 두 개의 특징을 같는 상수 집합으로 작성
- public static final 생략 특징을 활용!

❖ 문제점

- 다른 집합에 정의된 상수들은 서로 비교 하면 안됨
  (DAY.MONDAY == MONTH.JANUARY)
- 논리적 오류 이지만 컴파일 에러가 나지 않음
  (런타임 단계에서 예기치 못한 문제 발생할 수 있음!)

```
interface DAY {
    int MONDAY      = 1;
    int TUESDAY     = 2;
    int WEDNESDAY   = 3;
    int THURSDAY    = 4;
    int FRIDAY      = 5;
    int SATURDAY    = 6;
    int SUBDAY      = 7;
}

interface MONTH {
    int JANUARY   = 1;
    int FEBRUARY  = 2;
    int MARCH     = 3;
    int APRIL     = 4;
    int MAY       = 5;
    int JUNE      = 6;
    int JULY      = 7;
    int AUGUST    = 8;
    int SEPTEMBER = 9;
    int OCTOBER   = 10;
    int NOVEMBER  = 11;
    int DECEMBER  = 12;
}
```

enum의 필요성 (추가된 배경)

- 자기 자신의 객체를 할당!
  (서로 다른 데이터를 가지도록 성공!)
- 다른 상수 집합 끼리 **비교 불가** 하도록 성공!

❖ 문제점

- switch-case 문을 사용할 수 없게 됨 (if문은 가능)
- 거의 다 해결 했지만 뭔가 좀 아쉽다!

```java
class Day {
    public static final Day MONDAY    = new Day();
    public static final Day TUESDAY   = new Day();
    public static final Day WEDNESDAY = new Day();
    // ... 생략 ...
}
class Month {
    public static final Month JANUARY   = new Month();
    public static final Month FEBRUARY  = new Month();
    public static final Month MARCH     = new Month();
    // ... 생략...
}
public class Main {
    public static void main(String[] args) {
        if(Day.MONDAY == Month.JANUARY) {
            // 이제는 비교 할 수 없음!
        }

        Day day = Day.MONDAY;
        // switch - case 문제 사용할 수 없게 되었음...
        switch (day) {
        case 1:
            System.out.println("월요일 입니다.");
            break;
        case 2:
            System.out.println("화요일 입니다.");
            break;
            /*
             * ...생략...
             */
        }
    }
}
```

enum의 필요성

- 위와 같은 패턴이 자주 쓰이다 보니 문법적으로 지원함!
- **코드가 단순**해지며 **가독성**이 좋다!
- **객체**(인스턴스) **생성과 상속**을 방지!
- 키워드 enum을 사용하여 **의도가 열거임을 분명**히 할 수 있다!

```
class Day {
    public static final Day MONDAY    = new Day();
    public static final Day TUESDAY   = new Day();
    public static final Day WEDNESDAY = new Day();
}
```

```
enum Day {
    MONDAY, TUEDAY, WEDNESDAY;
}
```

enum에 멤버 추가

- 작성 및 사용법은 **일반 클래스에서 메서드 작성과 동일**하다.
- 주의! 상수들만 정의 시 ; **필수아님**
  하지만, **멤버를 추가하면 반드시 ;**

```
enum Greeting {
    GOOD_MORNING, GOOD_AFTERNOON, GOOD_EVENING;

    String message;
    public Greeting nextGreeting() {
        switch (this) {
        case GOOD_EVENING:      return GOOD_MORNING;
        case GOOD_MORNING:      return GOOD_AFTERNOON;
        default :               return GOOD_EVENING;
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Greeting curr = Greeting.GOOD_AFTERNOON;
        curr.message = "HELLO";
        Greeting next = curr.nextGreeting();
        next.message = "HI";
        System.out.println("지금: " + curr + ", 메시지: " + curr.message);
        System.out.println("다음: " + next + ", 메시지: " + next.message);

    }
}
```

enum과 생성자

- enum 클래스의 원소에 추가 속성 부여
- 생성자의 파라미터를 통해 추가 속성!
- enum 상수에 **어떤 데이터를 연관 시킬 수 있다!**

```
13 enum Season {
14     SPRING("봄"), SUMMER("여름"), FALL("가을"), WINTER("겨울");
15     private String season;
16     Season(String season) {
17         this.season = season;
18     }
19     public String getSeason() {
20         return this.season;
21     }
22 }
23 public class Main {
24     public static void main(String[] args) {
25         System.out.println(Season.SPRING.getSeason());
26         System.out.println(Season.SUMMER.getSeason());
27     }
28 }
```

Markers | Properties | Servers | Data Source Explorer | Snippets | Problems | Search | Console

<terminated> Main (1) [Java Application] C:₩Program Files₩Java₩jre1.8.0_301₩bin₩javaw.exe (2022. 1. 10. 오후 6:56

봄
여름

enum의 생성자

- enum 타입은 고정된 상수의 집합
- 즉, 런타임(run-time)이 아닌 **컴파일타임**(compile-time)에 **모든 값을 알고 있어야 함!**
- enum의 생성자는 항상 private (생략가능)
- 외부에서 접근 가능한 생성자가 없음 (final과 다름없다)

enum의 메서드

- enum 타입들은 java.lang.Enum 클래스를 기본적으로 상속받고 있기 때문에 Enum클래스에 선언된 메서드들 사용 가능

메서드 명	선언부와 설명
name()	enum상수의 이름을 문자열로 리턴한다.
ordinal()	0 부터 시작하는 enum상수의 순서를 리턴한다.
compareTo()	enum상수의 ordinal 차이를 리턴한다.
values()	Enum 타입에 선언된 enum 상수를 배열로 리턴한다.
valueOf()	문자열로 매핑 된 enum 상수 객체를 리턴한다.