

열거형, 가변 인자

25-1. 열거형

인터페이스 기반 상수의 정의: 자바 5 이전의 방식

```
interface Scale {  
    int DO = 0; int RE = 1; int MI = 2; int FA = 3;  
    int SO = 4; int RA = 5; int TI = 6;  
}
```

인터페이스 내에 선언된 변수는 `public, static, final`이 선언된 것으로 간주

이전 방식의 문제점

```
interface Animal {  
    int DOG = 1;  
    int CAT = 2;  
}
```

```
interface Person {  
    int MAN = 1;  
    int WOMAN = 2;  
}
```

```
class NonSafeConst {  
    public static void main(String[] args) {  
        who(Person.MAN);    // 정상적인 메소드 호출  
        who(Animal.DOG);    // 비정상적 메소드 호출  
    }  
}
```

컴파일 및 실행 과정에서 발견되지 않는 오류

```
    public static void who(int man) {  
        switch(man) {  
            case Person.MAN:  
                System.out.println("남성 손님입니다.");  
                break;  
            case Person.WOMAN:  
                System.out.println("여성 손님입니다.");  
                break;  
        }  
    }  
}
```

자료형의 부여를 돕는 열거형

```
enum Scale {      // 열거 자료형 Scale의 정의
```

```
    DO, RE, MI, FA, SO, RA, TI
```

```
}    열거형 값 (Enumerated Values)
```

case문에서는 표현의 간결함을 위해 Do와 같이
‘열거형 값’의 이름만 명시하기로 약속되어 있다.

```
public static void main(String[] args) {  
    Scale sc = Scale.DO;  
    System.out.println(sc);  
  
    switch(sc) {  
        case DO:  
            System.out.println("도~ ");  
            break;  
        case RE:  
            System.out.println("레~ ");  
            break;  
        case MI:  
            System.out.println("미~ ");  
            break;  
        case FA:  
            System.out.println("파~ ");  
            break;  
        default:  
            System.out.println("솔~ 라~ 시~ ");  
    }  
}
```

열거형 기반으로 수정한 결과와 개선된 부분

```
enum Animal {  
    DOG, CAT  
}
```

```
enum Person {  
    MAN, WOMAN  
}
```

```
class SafeEnum {  
    public static void main(String[] args) {  
        who(Person.MAN);    // 정상적인 메소드 호출  
        who(Animal.DOG);    // 비정상적 메소드 호출  
    }  
    컴파일 과정에서 자료형 불일치로 인한 오류 발생  
  
    public static void who(Person man) {  
        switch(man) {  
            case MAN:  
                System.out.println("남성 손님입니다.");  
                break;  
            case WOMAN:  
                System.out.println("여성 손님입니다.");  
                break;  
        }  
    }  
}
```

클래스 내에 열거형 정의 가능

```
class Customer {  
    enum Gender {          // 클래스 내에 정의된 열거형 Gender  
        MALE, FEMALE  
    }  
}
```

```
    private String name;  
    private Gender gen;
```

클래스 내에 열거형이 정의되면 해당 클래스 내에서
만 사용 가능한 열거형이 된다.

```
    Customer(String n, String g) {  
        name = n;  
  
        if(g.equals("man"))  
            gen = Gender.MALE;  
        else  
            gen = Gender.FEMALE;  
    }  
    . . .  
}
```

열거형 값의 정체: 이런 문장 삽입 가능합니다.

```
class Person {
    public static final Person MAN = new Person();
    public static final Person WOMAN = new Person();

    @Override
    public String toString() {
        return "I am a dog person";    // "나는 개를 사랑하는 사람입니다."
    }
}

class InClassInst {
    public static void main(String[] args) {
        System.out.println(Person.MAN);
        System.out.println(Person.WOMAN);
    }
}
```

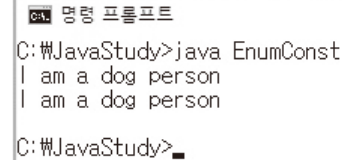

열거형 값의 정체: 열거형 값이 인스턴스라는 증거 1

```
enum Person {  
    MAN, WOMAN;  
  
    @Override  
    public String toString() { return "I am a dog person"; }  
}
```

```
class EnumConst {  
    public static void main(String[] args) {  
        System.out.println(Person.MAN);    // toString 메소드의 반환 값 출력  
        System.out.println(Person.WOMAN);    // toString 메소드의 반환 값 출력  
    }  
}
```

모든 열거형은 `java.lang.Enum<E>` 클래스를 상속한다.

그리고 `Enum<E>`는 `Object` 클래스를 상속한다. 이런 측면에서 볼 때 열거형은 클래스이다.



```
C:\JavaStudy>java EnumConst  
I am a dog person  
I am a dog person  
  
C:\JavaStudy>
```

열거형 값의 정체: 열거형 값이 인스턴스라는 증거2

```
enum Person {  
    MAN, WOMAN;
```

열거형의 정의에도 생성자가 없으면 디폴트 생성자가 삽입된다.

다만 이 생성자는 `private`으로 선언이 되어 직접 인스턴스를 생성하는 것이 불가능하다.

```
    private Person() {  
        System.out.println("Person constructor called");  
    }  
  
    @Override  
    public String toString() { return "I am a dog person"; }  
}  
  
class EnumConstructor {  
    public static void main(String[] args) {  
        System.out.println(Person.MAN);  
        System.out.println(Person.WOMAN);  
    }  
}
```

cmd 명령 프롬프트

```
C:\JavaStudy>java EnumConstructor  
Person constructor called  
Person constructor called  
I am a dog person  
I am a dog person  
C:\JavaStudy>
```

열거형 값의 정체: 결론

```
enum Person {  
    MAN, WOMAN;  
    . . . .  
}
```

```
public static final Person MAN = new Person();
```

```
public static final Person WOMAN = new  
Person();
```

열거형 값의 실체를 설명하는 문장 실제로 이렇게 컴파일이 되지는 않음

열거형 생성자에 인자 전달하기

```
enum Person {  
    MAN(29), WOMAN(25);  
  
    int age;  
    private Person(int age) {  
        this.age = age;  
    }  
  
    @Override  
    public String toString() {  
        return "I am " + age + " years old";  
    }  
}  
  
class EnumParamConstructor {  
    public static void main(String[] args) {  
        System.out.println(Person.MAN);  
        System.out.println(Person.WOMAN);  
    }  
}
```

cmd 명령 프롬프트

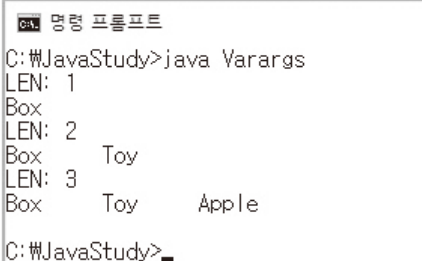
```
C:\JavaStudy>java EnumParamConstructor  
I am 29 years old  
I am 25 years old  
C:\JavaStudy>
```

25-2.

매개변수의 가변 인자 선언

매개변수의 가변 인자 선언과 호출

```
class Varargs {  
    public static void showAll(String...vargs) {  
        System.out.println("LEN: " + vargs.length);  
  
        for(String s : vargs)  
            System.out.print(s + '\t');  
        System.out.println();  
    }  
  
    public static void main(String[] args) {  
        showAll("Box");  
        showAll("Box", "Toy");  
        showAll("Box", "Toy", "Apple");  
    }  
}
```



```
명령 프롬프트  
C:\JavaStudy>java Varargs  
LEN: 1  
Box  
LEN: 2  
Box    Toy  
LEN: 3  
Box    Toy    Apple  
C:\JavaStudy>
```

가변 인자 선언에 대한 컴파일 처리

```
public static void showAll(String...vargs) {...}
```

vargs를 배열의 참조변수로 간주하고 코드를 작성하면 된다.

```
public static void main(String[] args) {  
    showAll("Box");  
    showAll("Box", "Toy");  
    showAll("Box", "Toy", "Apple");  
}
```



컴파일러가 다음과 같이 배열 기반 코드로 수정을 한다.

```
public static void showAll(String[] vargs) {...}
```

```
public static void main(String[] args) {  
    showAll(new String[]{"Box"});  
    showAll(new String[]{"Box", "Toy"});  
    showAll(new String[]{"Box", "Toy", "Apple"});  
}
```

25-3. 어노테이션

어노테이션의 설명 범위

`@Override`

`@Deprecated`

`@SuppressWarnings`

어노테이션 관련 문서

JSR 175 "A Metadata Facility for the Java Programming Language."

JSR 250 "Common Annotations for the Java Platform"

@Override

```
interface Viewable {  
    public void showIt(String str);  
}  
  
class Viewer implements Viewable {  
    @Override  
    public void showIt(String str) {  
        System.out.println(str);  
    }  
};
```

@Deprecated

```
interface Viewable {
    @Deprecated
    public void showIt(String str);

    public void brShowIt(String str);
}

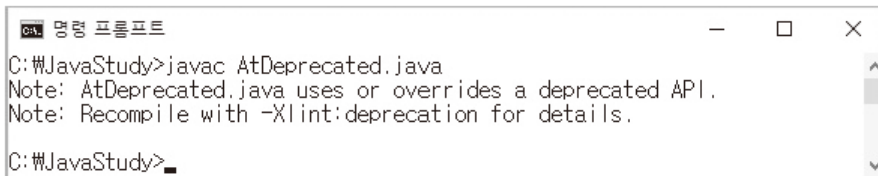
class Viewer implements Viewable {
    @Override
    public void showIt(String str) {
        System.out.println(str);
    }

    @Override
    public void brShowIt(String str) {
        System.out.println('[' + str + ']');
    }
}
```

// Deprecated 된 메소드

문제의 발생 소지가 있거나 개선된 기능의 다른 것으로 대체되어서 더 이상 필요 없게 되었음을 뜻 함

```
public static void main(String[] args) {
    Viewable view = new Viewer();
    view.showIt("Hello Annotations");
    ....
}
```



```
C:\JavaStudy>javac AtDeprecated.java
Note: AtDeprecated.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
C:\JavaStudy>
```

@SuppressWarnings

```
interface Viewable {
    @Deprecated
    public void showIt(String str);
    public void brShowIt(String str);
}

class Viewer implements Viewable {
    @Override
    @SuppressWarnings("deprecation")
    public void showIt(String str) { System.out.println(str); }

    @Override
    public void brShowIt(String str) { System.out.println('[' + str + ']'); }
};

class AtSuppressWarnings {
    @SuppressWarnings("deprecation")
    public static void main(String[] args) {
        Viewable view = new Viewer();
        view.showIt("Hello Annotations");
        view.brShowIt("Hello Annotations");
    }
}
```