

*Luke's Language*로 파싱된 이펙티브 자바

이념, 애노테이션.

제작. 홍성혁

보충 - Enum

그런데 어떤 규칙들은 너무 강력한 나머지
그 규칙을 지키지 않으면 그저 혼쭐이 나버리고



규칙에 대해 의문을 제기하는 것 자체도 어려울 때가 있다

보충 - Enum

Enumerated type

🌐 15 languages ▼

Article Talk

Read Edit View history Tools ▼

From Wikipedia, the free encyclopedia

In [computer programming](#), an **enumerated type** (also called **enumeration**, **enum**, or **factor** in the [R programming language](#), and a [categorical variable](#) in statistics) is a [data type](#) consisting of a set of named [values](#) called *elements*, *members*, *enumerals*, or *enumerators* of the type. The enumerator names are usually [identifiers](#) that behave as [constants](#) in the language. An enumerated type can be seen as a degenerate [tagged union](#) of [unit type](#). A [variable](#) that has been [declared](#) as having an enumerated type can be assigned any of the enumerators as a value. In other words, an enumerated type has values that are different from each other, and that can be compared and assigned, but are not specified by the programmer as having any particular concrete representation in the computer's memory; compilers and interpreters can represent them arbitrarily.



규칙에 대해 의문을 제기하는 것 자체도 어려울 때가 있다

@waterglasstoon

보충 - Enum

Enumerated type

📄 15 languages ▼

Article Talk

Read Edit View history Tools ▼

From Wikipedia, the free encyclopedia

컴퓨터 프로그래밍에서 열거형 (R 프로그래밍 언어에서는 열거형, 열거형 또는 요인 이라고도 하며 통계에서는 범주형 변수라고도 함)은 요소의 집합, 멤버, 열거형 또는 열거자로 구성된 데이터 유형입니다. 열거자 이름은 일반적으로 언어에서 상수로 작동하는 식별자입니다. 열거형은 단위 유형의 퇴화 태그 합집합으로 볼 수 있습니다. 열거형으로 선언된 변수에는 열거자 중 하나를 값으로 할당할 수 있습니다. 즉, 열거형은 서로 다른 값을 가지며 비교 및 할당할 수 있지만 프로그래머가 컴퓨터 메모리에 특정한 구체적인 표현이 있다고 지정하지 않습니다. 컴파일러와 인터프리터는 이를 임의로 표현할 수 있습니다.



규칙에 대해 의문을 제기하는 것 자체도 어려울 때가 있다

보충 - Enum in Java

Java [\[edit\]](#)

The J2SE version 5.0 of the [Java programming language](#) added enumerated types whose declaration syntax is similar to that of [C](#):

```
enum Cardsuit { CLUBS, DIAMONDS, SPADES, HEARTS };  
...  
Cardsuit trump;
```



The Java type system, however, treats enumerations as a type separate from integers, and intermixing of enum and integer values is not allowed. In fact, an enum type in Java is actually a special compiler-generated [class](#) rather than an arithmetic type, and enum values behave as global pre-generated instances of that class. Enum types can have instance methods and a constructor (the arguments of which can be specified separately for each enum value). All enum types implicitly extend the [Enum](#) [abstract class](#). An enum type cannot be instantiated directly.^[12]

Internally, each enum value contains an integer, corresponding to the order in which they are declared in the source code, starting from 0. The programmer cannot set a custom integer for an enum value directly, but one can define [overloaded constructors](#) that can then assign arbitrary values to self-defined members of the enum class. Defining getters allows then access to those self-defined members. The internal integer can be obtained from an enum value using the [ordinal\(\)](#) [method](#), and the list of enum values of an enumeration type can be obtained in order using the [values\(\)](#) [method](#). It is generally discouraged for programmers to convert enums to integers and vice versa.^[13] Enumerated types are [Comparable](#), using the internal integer; as a result, they can be sorted.

The Java standard library provides utility classes to use with enumerations. The [EnumSet](#) [class](#) implements a [Set](#) of enum values; it is implemented as a [bit array](#), which makes it very compact and as efficient as explicit bit manipulation, but safer. The [EnumMap](#) [class](#) implements a [Map](#) of enum values to object. It is implemented as an array, with the integer value of the enum value serving as the index.

보충 - Enum in Java

Java [edit]

The J2SE version 5.0 of the [Java programming language](#) added enumerated types whose declaration syntax is similar to that of C:

```
enum Cardsuit { CLUBS, DIAMONDS, SPADES, HEARTS };  
...  
Cardsuit trump;
```



그러나 Java 유형 시스템은 열거형을 정수와 별개의 유형으로 취급하며 열거형과 정수 값을 섞는 것은 허용되지 않습니다. 사실 Java의 열거형은 산술 유형이 아니라 컴파일러에서 생성한 특수 클래스이며 열거형 값은 해당 클래스의 전역 사전 생성 인스턴스처럼 작동합니다. 열거형 유형은 인스턴스 메서드와 생성자(각 열거형 값에 대해 별도로 지정할 수 있는 인수)를 가질 수 있습니다. 모든 열거형 유형은 [Enum](#) 추상 클래스를 암묵적으로 확장합니다. 열거형 유형은 직접 인스턴스화할 수 없습니다.^[12]

내부적으로 각 열거형 값은 소스 코드에서 선언된 순서에 따라 0부터 시작하는 정수를 포함합니다. 프로그래머는 열거형 값에 대한 사용자 정의 정수를 직접 설정할 수 없지만, 열거형 클래스의 자체 정의 멤버에 임의의 값을 할당할 수 있는 오버로드된 생성자를 정의할 수 있습니다. 게터를 정의하면 해당 자체 정의 멤버에 액세스할 수 있습니다. 내부 정수는 메서드를 사용하여 열거형 값에서 가져올 수 있으며 [ordinal\(\)](#), 열거형 유형의 열거형 값 목록은 메서드를 사용하여 순서대로 가져올 수 있습니다 [values\(\)](#). 일반적으로 프로그래머가 열거형을 정수로 변환하거나 그 반대로 변환하는 것은 권장되지 않습니다.^[13] 열거형은 [Comparable](#) 내부 정수를 사용하여 implements. 결과적으로 정렬할 수 있습니다.

Java 표준 라이브러리는 열거형과 함께 사용할 유틸리티 클래스를 제공합니다. 이 [EnumSet](#) 클래스는 Set 열거형 값의 a를 구현합니다. 비트 배열로 구현되어 매우 간결하고 명시적 비트 조작만큼 효율적이지만 더 안전합니다. 이 [EnumMap](#) 클래스는 객체에 대한 열거형 값의 a를 구현합니다. Map. 배열로 구현되며 열거형 값의 정수 값이 인덱스 역할을 합니다.

Item 37 - 말쑥한 EnumMap

ordinal 메서드

열거타입에서 몇 번째 위치인지 반환

Item 35에서 선언 순서 등의 사소한 변화에 취약하므로 쓰지 말라고 함

Item 37 - 말쑥한 EnumMap

EnumMap

```
Map<Plant.LifeCycle, Set<Plant>> plantsByLifeCycle =  
    new EnumMap<>(Plant.LifeCycle.class);
```


Item 37 - 말쑥한 EnumMap

EnumMap

```
Map<Plant.LifeCycle, Set<Plant>>  
new EnumMap<>(Plant.LifeCycle
```

```
public class EnumMap<K extends Enum<K>, V> extends AbstractMap<K, V>  
    implements java.io.Serializable, Cloneable
```

The `Class` object for the enum type of all the keys of this map.

```
private final Class<K> keyType;
```

All of the values comprising K. (Cached for performance.)

```
private transient K[] keyUniverse;
```

Array representation of this map. The *i*th element is the value to which `universe[i]` is currently mapped, or null if it isn't mapped to anything, or NULL if it's mapped to null.

```
private transient Object[] vals;
```

The number of mappings in this map.

```
private transient int size = 0;
```

Distinguished non-null value for representing null values.

```
private static final Object NULL = new Object() {  
    public int hashCode() { return 0; }  
  
    public String toString() { return "java.util.EnumMap.NULL"; }  
};
```

```
public class EnumMap<K extends Enum<K>, V> extends AbstractMap<K, V>
    implements java.io.Serializable, Cloneable
```

Item 37 - 말쑥한 EnumMap

Creates an empty enum map with the specified key type.

매개변수: `keyType` – the class object of the key type for this enum map

던지기: `NullPointerException` – if `keyType` is null

```
public EnumMap(Class<K> keyType) {
    this.keyType = keyType;
    keyUniverse = getKeyUniverse(keyType);
    vals = new Object[keyUniverse.length];
}
```

The `Class` object for the enum type of all the keys of this map.

```
final Class<K> keyType;
```

values comprising K. (Cached for performance.)

```
transient K[] keyUniverse;
```

representation of this map. The *i*th element is the value to which `universe[i]` is currently mapped, or null if it isn't mapped to anything, or NULL if it's mapped to null.

```
transient Object[] vals;
```

number of mappings in this map.

```
private transient int size = 0;
```

Distinguished non-null value for representing null values.

```
private static final Object NULL = new Object() {
    public int hashCode() { return 0; }

    public String toString() { return "java.util.EnumMap.NULL"; }
};
```

```
public class EnumMap<K extends Enum<K>, V> extends AbstractMap<K, V>
    implements java.io.Serializable, Cloneable
```

Item 37 - 말쑥한 EnumMap

The `Class` object for the enum type of all the keys of this map.

Creates an empty enum map with the specified key type.

매개변수: `keyType` – the class object of the key type for this enum map

```
final Class<K> keyType;
```

던지기: Creates an enum map with the same key type as the specified enum map, initially containing the same mappings (if any).

매개변수: `m` – the enum map from which to initialize this enum map

던지기: `NullPointerException` – if `m` is null

```
public EnumMap(Class<K> keyType) {
    this(keyType, EnumSet.allOf(keyType));
}

public EnumMap(EnumMap<K, ? extends V> m) {
    keyType = m.keyType;
    keyUniverse = m.keyUniverse;
    vals = m.vals.clone();
    size = m.size;
}

public String toString() { return "java.util.EnumMap.NULL"; }
```

due to which universe[i] is currently mapped to null.

```
public String toString() { return "java.util.EnumMap.NULL"; }
};
```

Item 37 -

Creates an empty enum map.

매개변수: `keyType` -

던지기: Creates an empty enum map with the same mapping as the specified map.

```
public EnumMap<K, V> of()  
{  
    this.  
    keyUniverse =  
    vals =  
}
```

매개변수: `m`

던지기: `m`

```
public EnumMap<K, V> ofAll(Map<K, V> m)  
{  
    keyType = m.keySet().iterator().next().getDeclaringClass();  
    keyUniverse = getKeyUniverse(keyType);  
    vals = new Object[keyUniverse.length];  
    size = 0;  
    putAll(m);  
}
```

Creates an enum map initialized from the specified map. If the specified map is an `EnumMap` instance, this constructor behaves identically to `EnumMap(EnumMap)`. Otherwise, the specified map must contain at least one mapping (in order to determine the new enum map's key type).

매개변수: `m` - the map from which to initialize this enum map

던지기: `IllegalArgumentException` - if `m` is not an `EnumMap` instance and contains no mappings

`NullPointerException` - if `m` is null

```
public EnumMap<K, ? extends V> m) {  
    if (m instanceof EnumMap) {  
        EnumMap<K, ? extends V> em = (EnumMap<K, ? extends V>) m;  
        keyType = em.keyType;  
        keyUniverse = em.keyUniverse;  
        vals = em.vals.clone();  
        size = em.size;  
    } else {  
        if (m.isEmpty())  
            throw new IllegalArgumentException("Specified map is empty");  
        keyType = m.keySet().iterator().next().getDeclaringClass();  
        keyUniverse = getKeyUniverse(keyType);  
        vals = new Object[keyUniverse.length];  
        putAll(m);  
    }  
}
```

EnumMap<K, V>

Each universe[i] is currently
d to null.

map.NULL"; }

Item 37 - 말쑥한 EnumMap

EnumMap을 이용한 열거

코드 37-2 EnumMap을 사용해 데이터와 열거 타입을 매핑한다.

```
Map<Plant.LifeCycle, Set<Plant>> plantsByLifeCycle =  
    new EnumMap<>(Plant.LifeCycle.class);  
for (Plant.LifeCycle lc : Plant.LifeCycle.values())  
    plantsByLifeCycle.put(lc, new HashSet<>());  
for (Plant p : garden)  
    plantsByLifeCycle.get(p.lifeCycle).add(p);  
System.out.println(plantsByLifeCycle);
```

Item 37 - 말쑥한 EnumMap

EnumMap을 이용한 그룹핑

코드 37-4 스트림을 사용한 코드 2 - EnumMap을 이용해 데이터와 열거 타입을 매핑했다.

```
System.out.println(Arrays.stream(garden)
    .collect(groupingBy(p -> p.lifeCycle,
        () -> new EnumMap<>(LifeCycle.class), toSet())));
```


Item 37 - 말쑥한 EnumMap

ordinal vs EnumMap 비교

코드 37-5 배열들의 배열의 인덱스에 ordinal()을 사용 - 따라 하지 말 것!

```
public enum Phase {  
    SOLID, LIQUID, GAS;  
  
    public enum Transition {  
        MELT, FREEZE, BOIL, CONDENSE, SUBLIME, DEPOSIT;  
  
        // 행은 from의 ordinal을, 열은 to의 ordinal을 인덱스로 쓴다.  
        private static final Transition[][] TRANSITIONS = {  
            { null, MELT, SUBLIME },  
            { FREEZE, null, BOIL },  
            { DEPOSIT, CONDENSE, null }  
        };  
  
        // 한 상태에서 다른 상태로의 전이를 반환한다.  
        public static Transition from(Phase from, Phase to) {  
            return TRANSITIONS[from.ordinal()][to.ordinal()];  
        }  
    }  
}
```

코드 37-5 배열들의 배열의 인덱스에 ordinal()을 사용 - 따라 하지 말 것!

```
public enum Phase {
    SOLID, LIQUID, GAS;

    public enum Transition {
        MELT, FREEZE, BOIL, CONDENSE, SUBLIME, DEPOSIT;

        // 행은 from의 ordinal을, 열은 to의 ordinal을 인덱스로 쓴다.
        private static final Transition[][] TRANSITIONS = {
            { null, MELT, SUBLIME },
            { FREEZE, null, BOIL },
            { DEPOSIT, CONDENSE, null }
        };

        // 한 상태에서 다른 상태로의 전이를 반환한다.
        public static Transition from(Phase from, Phase to) {
            return TRANSITIONS[from.ordinal()][to.ordinal()];
        }
    }
}
```

Item 37 - 말쑥한 EnumMap

ordinal vs EnumMap 비교

=> Exception 유발

ArrayIndexOutOfBoundsException, NullPointerException

=> 새로운 Transition 추가가 어려움

Item 37 - 말쑥한 EnumMap

ordinal vs EnumMap 비교

코드 37-6 중첩 EnumMap으로 데이터와 열거 타입 쌍을 연결했다.

```
public enum Phase {  
    SOLID, LIQUID, GAS;  
  
    public enum Transition {  
        MELT(SOLID, LIQUID), FREEZE(LIQUID, SOLID),  
        BOIL(LIQUID, GAS), CONDENSE(GAS, LIQUID),  
        SUBLIME(SOLID, GAS), DEPOSIT(GAS, SOLID);  
  
        private final Phase from;  
        private final Phase to;  
  
        Transition(Phase from, Phase to) {  
            this.from = from;  
            this.to = to;  
        }  
  
        // 상전이 맵을 초기화한다.  
        private static final Map<Phase, Map<Phase, Transition>>  
            m = Stream.of(values()).collect(groupingBy(t -> t.from,  
                () -> new EnumMap<>(Phase.class),  
                toMap(t -> t.to, t -> t,  
                    (x, y) -> y, () -> new EnumMap<>(Phase.class))));  
  
        public static Transition from(Phase from, Phase to) {  
            return m.get(from).get(to);  
        }  
    }  
}
```

Item 37 - 말쑥한 EnumMap

ordinal vs EnumMap 비교

코드 37-6 중첩 EnumMap으로 데이터와 열거 타입 쌍을 연결했다.

```
public enum Phase {  
    SOLID, LIQUID, GAS;  
  
    public enum Transition {  
        MELT(SOLID, LIQUID), FREEZE(LIQUID, SOLID),  
        BOIL(LIQUID, GAS), CONDENSE(GAS, LIQUID),  
        SUBLIME(SOLID, GAS), DEPOSIT(GAS, SOLID);  
  
        private final Phase from;  
        private final Phase to;  
  
        Transition(Phase from, Phase to) {  
            this.from = from;  
            this.to = to;  
        }  
  
        // 상전이 맵을 초기화한다.  
        private static final Map<Phase, Map<Phase, Transition>>  
            m = Stream.of(values()).collect(groupingBy(t -> t.from,  
                () -> new EnumMap<>(Phase.class),  
                toMap(t -> t.to, t -> t,  
                    (x, y) -> y, () -> new EnumMap<>(Phase.class))));  
  
        public static Transition from(Phase from, Phase to) {  
            return m.get(from).get(to);  
        }  
    }  
}
```

Item 38 - Enum의 Polymorphism

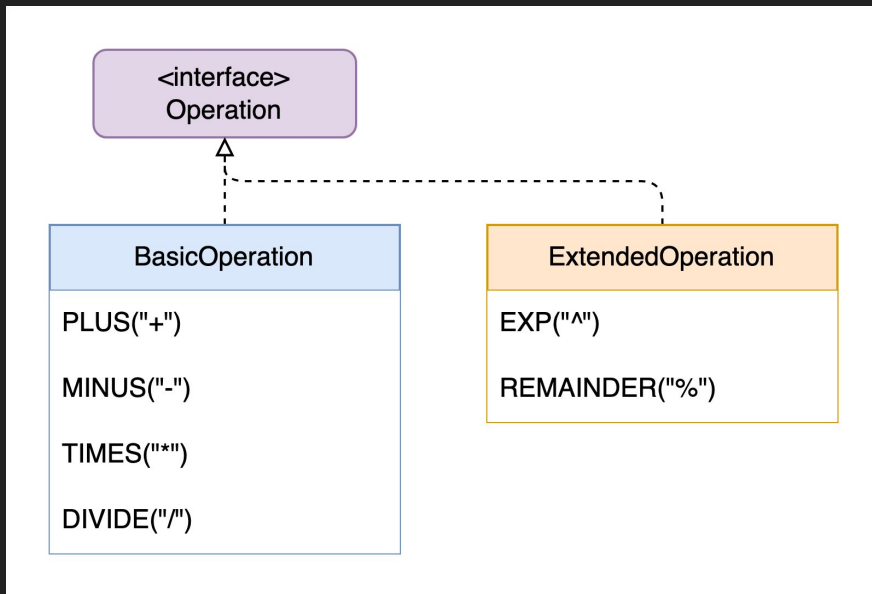
enum은 확장할 수 없다.

Item 38 - Enum의 Polymorphism

enum은 확장할 수 없다.
enum은 static한 클래스다.

Item 38 - Enum의 Polymorphism

enum은 확장할 수 없다.
enum은 static한 클래스다.
interface 구현은 가능

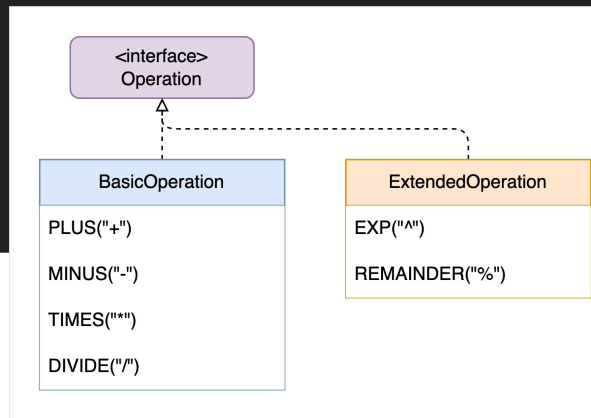


Item 38 - Enum의 Polymorphism

```
public interface Operation {  
    double apply(double x, double y);  
}
```

```
public enum BasicOperation implements Operation {  
    PLUS("+") {  
        public double apply(double x, double y) { return x + y; }  
    },  
    MINUS("-") {  
        public double apply(double x, double y) { return x - y; }  
    },  
    TIMES("*") {  
        public double apply(double x, double y) { return x * y; }  
    },  
    DIVIDE("/") {  
        public double apply(double x, double y) { return x / y; }  
    };  
  
    private final String symbol;  
  
    BasicOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

```
public enum ExtendedOperation implements Operation {  
    EXP("^") {  
        public double apply(double x, double y) {  
            return Math.pow(x, y);  
        }  
    },  
    REMAINDER("%") {  
        public double apply(double x, double y) {  
            return x % y;  
        }  
    };  
  
    private final String symbol;  
  
    ExtendedOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

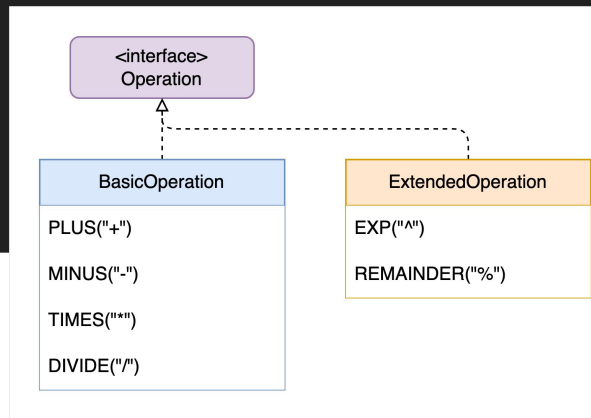


Item 38 - Enum의 Polymorphism

```
public interface Operation {  
    double apply(double x, double y);  
}
```

```
public enum BasicOperation implements Operation {  
    PLUS("+") {  
        public double apply(double x, double y) { return x + y; }  
    },  
    MINUS("-") {  
        public double apply(double x, double y) { return x - y; }  
    },  
    TIMES("*") {  
        public double apply(double x, double y) { return x * y; }  
    },  
    DIVIDE("/") {  
        public double apply(double x, double y) { return x / y; }  
    };  
  
    private final String symbol;  
  
    BasicOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

```
public enum ExtendedOperation implements Operation {  
    EXP("^") {  
        public double apply(double x, double y) {  
            return Math.pow(x, y);  
        }  
    },  
    REMAINDER("%") {  
        public double apply(double x, double y) {  
            return x % y;  
        }  
    };  
  
    private final String symbol;  
  
    ExtendedOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

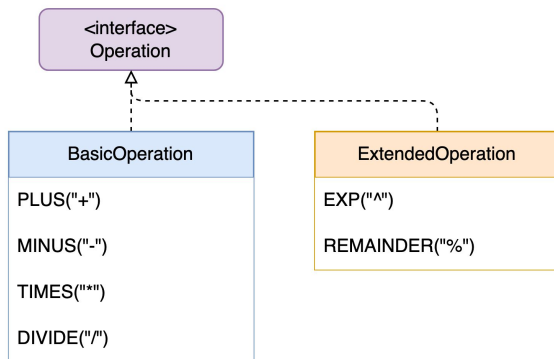


Item 38 - Enum의 Polymorphism

```
public interface Operation {  
    double apply(double x, double y);  
}
```

```
public enum BasicOperation implements Operation {  
    PLUS("+") {  
        public double apply(double x, double y) { return x + y; }  
    },  
    MINUS("-") {  
        public double apply(double x, double y) { return x - y; }  
    },  
    TIMES("*") {  
        public double apply(double x, double y) { return x * y; }  
    },  
    DIVIDE("/") {  
        public double apply(double x, double y) { return x / y; }  
    };  
  
    private final String symbol;  
  
    BasicOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

```
public enum ExtendedOperation implements Operation {  
    EXP("^") {  
        public double apply(double x, double y) {  
            return Math.pow(x, y);  
        }  
    },  
    REMAINDER("%") {  
        public double apply(double x, double y) {  
            return x % y;  
        }  
    };  
  
    private final String symbol;  
  
    ExtendedOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

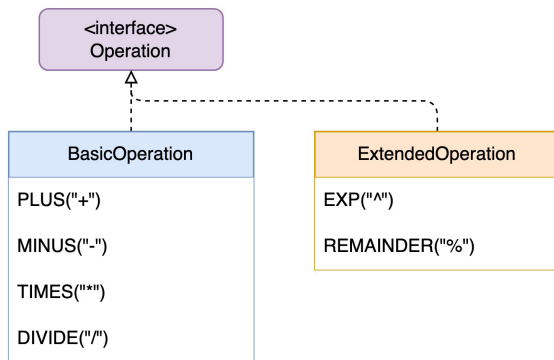


Item 38 - Enum의 Polymorphism

```
public interface Operation {  
    double apply(double x, double y);  
}
```

```
public enum BasicOperation implements Operation {  
    PLUS("+") {  
        public double apply(double x, double y) { return x + y; }  
    },  
    MINUS("-") {  
        public double apply(double x, double y) { return x - y; }  
    },  
    TIMES("*") {  
        public double apply(double x, double y) { return x * y; }  
    },  
    DIVIDE("/") {  
        public double apply(double x, double y) { return x / y; }  
    };  
  
    private final String symbol;  
  
    BasicOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```

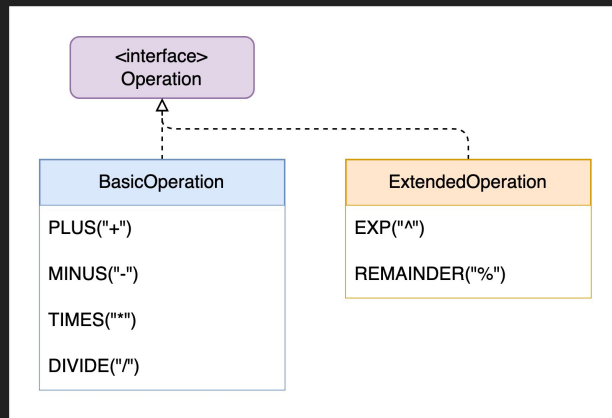
```
public enum ExtendedOperation implements Operation {  
    EXP("^") {  
        public double apply(double x, double y) {  
            return Math.pow(x, y);  
        }  
    },  
    REMAINDER("%") {  
        public double apply(double x, double y) {  
            return x % y;  
        }  
    };  
  
    private final String symbol;  
  
    ExtendedOperation(String symbol) {  
        this.symbol = symbol;  
    }  
  
    @Override public String toString() {  
        return symbol;  
    }  
}
```



Item 38 - Enum의 Polymorphism

```
public static void main(String[] args) {  
    double x = Double.parseDouble(args[0]);  
    double y = Double.parseDouble(args[1]);  
    test(ExtendedOperation.class, x, y);  
}  
  
private static <T extends Enum<T> & Operation> void test(  
    Class<T> opEnumType, double x, double y) {  
    for (Operation op : opEnumType.getEnumConstants())  
        System.out.printf("%f %s %f = %f%n",  
            x, op, y, op.apply(x, y));  
}
```

=> T는 Enum<T> 와 Operation을 상속



보충 - Annotation

Java annotation

文 13 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

Java 컴퓨터 [프로그래밍 언어](#) 에서 주석은 Java [소스 코드](#) 에 추가할 수 있는 구문 [메타데이터](#) 의 한 형태입니다. ^[1] 클래스, 메서드, 변수, 매개변수 및 [Java 패키지](#) 에 주석을 달 수 있습니다. [Javadoc](#) 태그와 마찬가지로 Java 주석은 소스 파일에서 읽을 수 있습니다. [Javadoc](#) 태그와 달리 Java 주석은 [Java 컴파일러](#) 에서 생성한 [Java 클래스 파일](#) 에 임베드하여 읽을 수도 있습니다. 이를 통해 주석은 런타임에 [Java 가상 머신](#) 에서 보관되고 리플렉션을 통해 읽을 수 있습니다. ^[2] Java에서 기존 주석에서 메타 주석을 만들 수 있습니다. ^[3]

보충 - Annotation

Java annotation

文 13 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

Java 컴퓨터 프로그래밍 언어 에서 주석은 Java 소스 코드 에 추가할 수 있는 구문 메타데이터 의 한 형태입니다. ^[1] 클래스, 메서드, 변수, 매개변수 및 Java 패키지 에 주석을 달 수 있습니다. Javadoc 태그와 마찬가지로 Java 주석은 소스 파일에서 읽을 수 있습니다 Javadoc 태그와 달리 Java 주석은 Java 컴파일러 에서 생성한 Java 클래스 파일 에 임베드하여 읽을 수도 있습니다. 이를 통해 주석은 런타임 에 Java 가상 머신 에서 보관되고 리플렉션을 통해 읽을 수 있습니다. ^[2] Java에서 기존 주석에서 메타 주석을 만들 수 있습니다. ^[3]

보충 - Annotation

Java annotation

文 13 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

Java 컴퓨터 프로그래밍 언어에서 주석은 Java 소스 코드에 추가할 수 있는 구문 메타데이터의 한 형태입니다.^[1] 클래스, 메서드, 변수, 매개변수 및 Java 패키지 에 주석을 달 수 있습니다. Javadoc 태그와 마찬가지로 Java 주석은 소스 파일에서 읽을 수 있습니다. Javadoc 태그와 달리 Java 주석은 Java 컴파일러에서 생성한 Java 클래스 파일에 임베드하여 읽을 수도 있습니다. 이를 통해 주석은 런타임에 Java 가상 머신에서 보관되고 리플렉션을 통해 읽을 수 있습니다.^[2] Java에서 기존 주석에서 메타 주석을 만들 수 있습니다.^[3]

보충 - Annotation

Java annotation

文 13 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

Java 컴퓨터 프로그래밍 언어 에서 주석은 Java 소스 코드 에 추가할 수 있는 구문 메타데이터 의 한 형태입니다. ^[1] 클래스, 메서드, 변수, 매개변수 및 Java 패키지 에 주석을 달 수 있습니다. Javadoc 태그와 마찬가지로 Java 주석은 소스 파일에서 읽을 수 있습니다. Javadoc 태그와 달리 Java 주석은 Java 컴파일러 에서 생성한 Java 클래스 파일 에 임베드하여 읽을 수도 있습니다. 이를 통해 주석은 런타임 에 Java 가상 머신 에서 보관되고 리플렉션을 통해 읽을 수 있습니다. ^[2] Java에서 기존 주석에서 메타 주석을 만들 수 있습니다. ^[3]

Item 39 - Annotation, 어떻게 쓸 것인가

코드 39-1 마커(marker) 애너테이션 타입 선언

```
import java.lang.annotation.*;

/**
 * 테스트 메서드임을 선언하는 애너테이션이다.
 * 매개변수 없는 정적 메서드 전용이다.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
}
```

Item 39 - Annotation, 어떻게 쓸 것인가

코드 39-1 마커(marker) 애너테이션 타입 선언

```
import java.lang.annotation.*;

/**
 * 테스트 메서드임을 선언하는 애너테이션이다.
 * 매개변수 없는 정적 메서드 전용이다.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
}
```

=> RUNTIME에도 유지되게 함

Item 39 - Annotation, 어떻게 쓸 것인가

코드 39-1 마커(marker) 애너테이션 타입 선언

```
import java.lang.annotation.*;

/**
 * 테스트 메서드임을 선언하는 애너테이션이다.
 * 매개변수 없는 정적 메서드 전용이다.
 */
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {
}
```

Annotation에는 적절한 애너테이션 처리기가 필요

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용1 - 배열 매개변수

코드 39-7 배열 매개변수를 받는 애너테이션을 사용하는 코드

```
@ExceptionTest({ IndexOutOfBoundsException.class,
                  NullPointerException.class })
public static void doublyBad() { // 성공해야 한다.
    List<String> list = new ArrayList<>();

    // 자바 API 명세에 따르면 다음 메서드는 IndexOutOfBoundsException이나
    // NullPointerException을 던질 수 있다.
    list.addAll(5, null);
}
```

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용1 - 배열 매개변수

코드 39-7 배열 매개변수를 받는 애너테이션을 사용하는 코드

```
@ExceptionTest { IndexOutOfBoundsException.class,  
                  NullPointerException.class }  
public static void doublyBad() { // 성공해야 한다.  
    List<String> list = new ArrayList<>();  
  
    // 자바 API 명세에 따르면 다음 메서드는 IndexOutOfBoundsException이나  
    // NullPointerException을 던질 수 있다.  
    list.addAll(5, null);  
}
```

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용1 - 배열 매개변수

코드 39-7 배열 매개변수를 받는 애너테이션을 사용하는 코드

```
@ExceptionTest({ IndexOutOfBoundsException.class,
                  NullPointerException.class })
public static void doublyBad() { // 성공해야 한다.
    List<String> list = new ArrayList<>();

    // 자바 API 명세에 따르면 다음 메서드는 IndexOutOfBoundsException이나
    // NullPointerException을 던질 수 있다.
    list.addAll(5, null);
}
```

코드 39-6 배열 매개변수를 받는 애너테이션 타입

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Throwable>[] value();
}
```

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용1 - 배열 매개변수

코드 39-7 배열 매개변수를 받는 애너테이션을 사용하는 코드

```
@ExceptionTest({ IndexOutOfBoundsException.class,
                  NullPointerException.class })
public static void doublyBad() { // 성공해야 한다.
    List<String> list = new ArrayList<>();

    // 자바 API 명세에 따르면 다음 메서드는 IndexOutOfBoundsException이나
    // NullPointerException을 던질 수 있다.
    list.addAll(5, null);
}
```

코드 39-6 배열 매개변수를 받는 애너테이션 타입

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Throwable>[] value();
}
```

=> 배열을 value 매개변수로 받는 애너테이션

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용1 - 배열 매개변수

코드 39-7 배열 매개변수를 받는 애너테이션을 사용하는 코드

```
@ExceptionTest({ IndexOutOfBoundsException.class,
                  NullPointerException.class })
public static void doublyBad() { // 성공해야 한다.
    List<String> list = new ArrayList<>();

    // 자바 API 명세에 따르면 다음 메서드는 IndexOutOfBoundsException이나
    // NullPointerException을 던질 수 있다.
    list.addAll(5, null);
}
```

코드 39-6 배열 매개변수를 받는 애너테이션 타입

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Throwable>[] value();
}
```

```
if (m.isAnnotationPresent(ExceptionTest.class)) {
    tests++;
    try {
        m.invoke(null);
        System.out.printf("테스트 %s 실패: 예외를 던지지 않음%n", m);
    } catch (Throwable wrappedExc) {
        Throwable exc = wrappedExc.getCause();
        int oldPassed = passed;
        Class<? extends Throwable>[] excTypes =
            m.getAnnotation(ExceptionTest.class).value();
        for (Class<? extends Throwable> excType : excTypes) {
            if (excType.isInstance(exc)) {
                passed++;
                break;
            }
        }
        if (passed == oldPassed)
            System.out.printf("테스트 %s 실패: %s %n", m, exc);
    }
}
```

=> 배열을 처리하는 애너테이션 처리기

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용1 - 배열 매개변수

코드 39-7 배열 매개변수를 받는 애너테이션을 사용하는 코드

```
@ExceptionTest({ IndexOutOfBoundsException.class,
                  NullPointerException.class })
public static void doublyBad() { // 성공해야 한다.
    List<String> list = new ArrayList<>();

    // 자바 API 명세에 따르면 다음 메서드는 IndexOutOfBoundsException이나
    // NullPointerException을 던질 수 있다.
    list.addAll(5, null);
}
```

코드 39-6 배열 매개변수를 받는 애너테이션 타입

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTest {
    Class<? extends Throwable>[] value();
}
```

```
if (m.isAnnotationPresent(ExceptionTest.class)) {
    tests++;
    try {
        m.invoke(null);
        System.out.printf("테스트 %s 실패: 예외를 던지지 않음%n", m);
    } catch (Throwable wrappedExc) {
        Throwable exc = wrappedExc.getCause();
        int oldPassed = passed;
        Class<? extends Throwable>[] excTypes =
            m.getAnnotation(ExceptionTest.class).value();
        for (Class<? extends Throwable> excType : excTypes) {
            if (excType.isInstance(exc)) {
                passed++;
                break;
            }
        }
        if (passed == oldPassed)
            System.out.printf("테스트 %s 실패: %s %n", m, exc);
    }
}
```

=> 배열을 처리하는 애너테이션 처리기

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionTest(IndexOutOfBoundsException.class)
@ExceptionTest(NullPointerException.class)
public static void doublyBad() { ... }
```

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionHandler(IndexOutOfBoundsException.class)  
@ExceptionHandler(NullPointerException.class)  
public static void doublyBad() { ... }
```

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionTest(IndexOutOfBoundsException.class)
@ExceptionTest(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionTest(IndexOutOfBoundsException.class)
@ExceptionTest(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```


=> 반복 가능 애너테이션을 만들기,
@Repeatable

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionTest(IndexOutOfBoundsException.class)
@ExceptionTest(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```



```
// 컨테이너 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTestContainer {
    ExceptionTest[] value();
}
```

=> 반복 가능 애너테이션을 만들기,

@Repeatable

1. 컨테이너 애너테이션이 필요하다.

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionHandler(IndexOutOfBoundsException.class)
@ExceptionHandler(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```

```
// 컨테이너 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTestContainer {
    ExceptionTest[] value();
}
```

```
private static <A extends Annotation> A[] getIndirectlyPresent(
    Map<Class<? extends Annotation>, Annotation> annotations,
    Class<A> annoClass) {

    Repeatable repeatable = annoClass.getDeclaredAnnotation(Repeatable.class);
    if (repeatable == null)
        return null; // Not repeatable -> no indirectly present annotations

    Class<? extends Annotation> containerClass = repeatable.value();

    Annotation container = annotations.get(containerClass);
    if (container == null)
        return null;

    // Unpack container
    A[] valueArray = getValueArray(container);
    checkTypes(valueArray, container, annoClass);

    return valueArray;
}
```

=> @Repeatable이 처리되는 과정

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionHandler(IndexOutOfBoundsException.class)
@ExceptionHandler(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```

```
// 컨테이너 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTestContainer {
    ExceptionTest[] value();
}
```

```
private static <A extends Annotation> A[] getIndirectlyPresent(
    Map<Class<? extends Annotation>, Annotation> annotations,
    Class<A> annoClass) {

    Repeatable repeatable = annoClass.getDeclaredAnnotation(Repeatable.class);
    if (repeatable == null)
        return null; // Not repeatable -> no indirectly present annotations

    Class<? extends Annotation> containerClass = repeatable.value();
    Annotation container = annotations.get(containerClass);
    if (container == null)
        return null;

    // Unpack container
    A[] valueArray = getValueArray(container);
    checkTypes(valueArray, container, annoClass);

    return valueArray;
}
```

=> @Repeatable이 처리되는 과정
Annotation container를 뽑아낸다.

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서 달기

```
@ExceptionHandler(IndexOutOfBoundsException.class)
@ExceptionHandler(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```

```
// 컨테이너 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTestContainer {
    ExceptionTest[] value();
}
```

=> 반복 가능 애너테이션을 만들기, **@Repeatable**

1. 컨테이너 애너테이션이 필요하다.
2. 내부 애너테이션 타입의 배열 **value** 를 정의한다.

Item 39 - Annotation, 어떻게 쓸 것인가

애너테이션 응용2 - 애너테이션 반복해서

```
@ExceptionTest(IndexOutOfBoundsException.class)
@ExceptionTest(NullPointerException.class)
public static void doublyBad() { ... }
```

```
// 반복 가능한 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@Repeatable(ExceptionTestContainer.class)
public @interface ExceptionTest {
    Class<? extends Throwable> value();
}
```

```
// 컨테이너 애너테이션
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ExceptionTestContainer {
    ExceptionTest[] value();
}
```

```
if (m.isAnnotationPresent(ExceptionTest.class)
    || m.isAnnotationPresent(ExceptionTestContainer.class))
    tests++;
try {
    m.invoke(null);
    System.out.printf("테스트 %s 실패: 예외를 던지지 않음%n", m);
} catch (Throwable wrappedExc) {
    Throwable exc = wrappedExc.getCause();
    int oldPassed = passed;
    ExceptionTest[] excTests =
        m.getAnnotationsByType(ExceptionTest.class);
    for (ExceptionTest excTest : excTests) {
        if (excTest.value().isInstance(exc)) {
            passed++;
            break;
        }
    }
    if (passed == oldPassed)
        System.out.printf("테스트 %s 실패: %s %n", m, exc);
}
```

=> 반복 애너테이션 처리기

E.O.D.