Effective Java

Item 35. ordinal 메서드 대신 인스턴스 필드를 사용하라.

개요

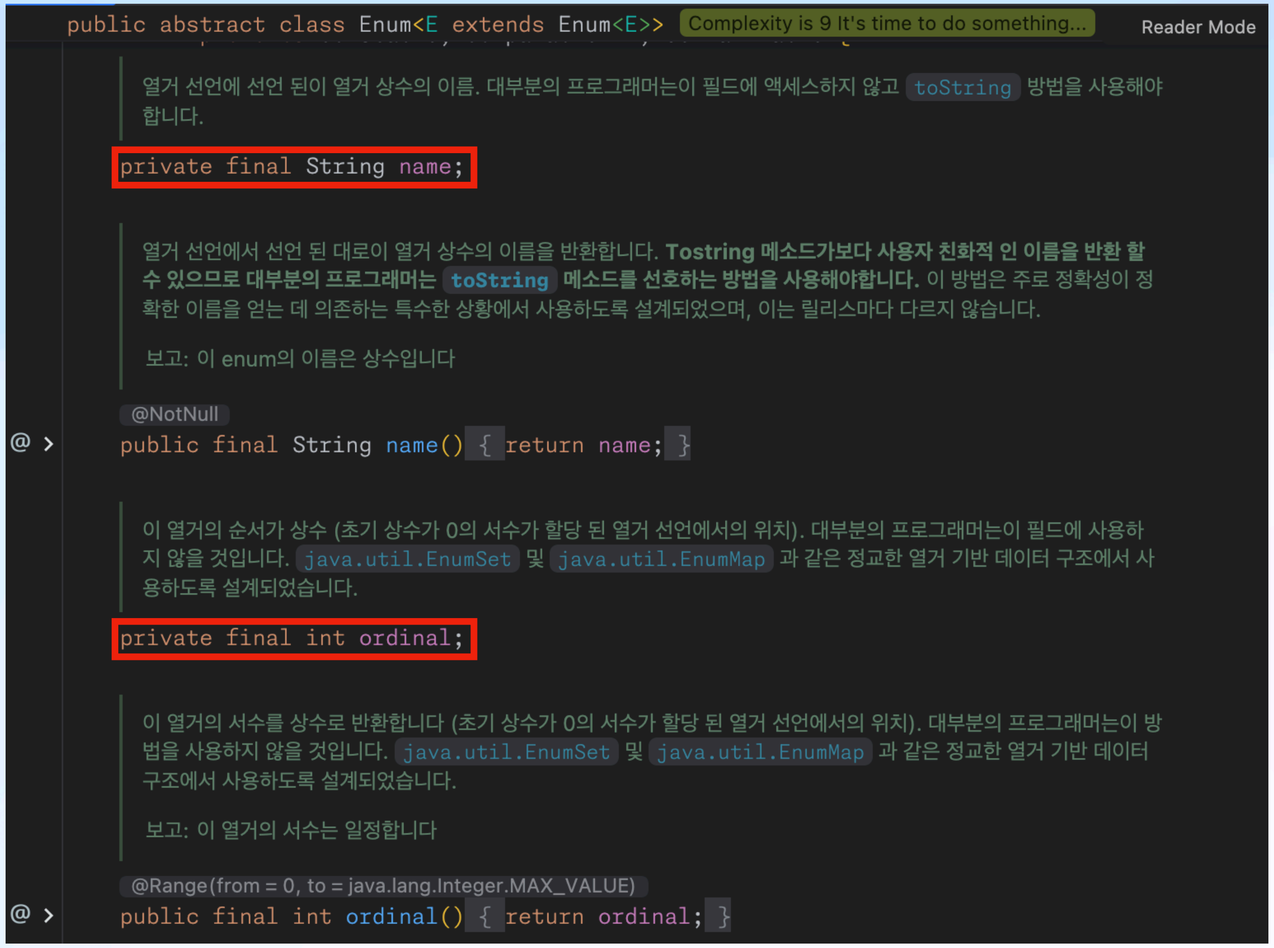
1. ordinal() 존재 이유
2. ordinal() 사용 단점
3. ordinal() 대안
4. Summary

개요

1. ordinal() 존재 이유
2. ordinal() 사용 단점
3. ordinal() 대안
4. Summary

ordinal() 존재 이유

java.lang.Enum : 모든 enum 타입의 부모 클래스



ordinal() 존재 이유

java.lang.Enum : 모든 enum 타입의 부모 클래스

> 이 열거의 서수를 상수로 반환합니다 (초기 상수가 0의 서수가 할당 된 열거 선언에서의 위치). 대부분의 프로그래머는이 방법을 사용하지 않을 것입니다. `java.util.EnumSet` 및 `java.util.EnumMap` 과 같은 정교한 열거 기반 데이터 구조에서 사용하도록 설계되었습니다.
>
> 보고: 이 열거의 서수는 일정합니다

```
@Range(from = 0, to = java.lang.Integer.MAX_VALUE)
public final int ordinal() { return ordinal; }
```

ordinal() 존재 이유 : EnumSet

EnumSet : Enum 타입에 최적화된 Set

```
public abstract sealed class EnumSet<E extends Enum<E>> extends AbstractSet<E>  Complexity
        implements Cloneable, java.io.Serializable permits JumboEnumSet, RegularEnumSet
{
```

```java
public enum Team {

    HAN("한"),
    CHO("초"),
    NONE("해당없음");

    public static final Set<Team> TEAMS = EnumSet.of(HAN, CHO);
    // public static final Set<Team> TEAMS = EnumSet.complementOf(EnumSet.of(NONE));
}
```

ordinal() 존재 이유 : EnumSet

EnumSet : Enum 타입에 최적화된 Set

```
final class RegularEnumSet<E extends Enum<E>> extends EnumSet<E> {  Complexity is 14 You must be kidding
    @java.io.Serial
    private static final long serialVersionUID = 3411599620347842686L;

    이 세트의 비트 벡터 표현. 2^k 비트는이 세트에서 우주 [k]의 존재를 나타냅니다.

    private long elements = 0L;
```

ordinal() 존재 이유 : EnumSet

```
이 세트에 지정된 요소가 포함 된 경우 true 반환합니다.

매개 변수 : e -이 컬렉션의 격리를 확인할 요소

보고:       이 세트에 지정된 요소가 포함 된 경우 true

public boolean contains(Object e) {  Complexity is 13 You must be kidding
    if (e == null)
        return false;
    Class<?> eClass = e.getClass();
    if (eClass != elementType && eClass.getSuperclass() != elementType)
        return false;

    return (elements & (1L << ((Enum<?>) e).ordinal())) != 0;
}
```

O(1) 시간복잡도로 포함 여부 조회

```
1  public enum Team {
2
3      HAN("한"), // ordinal = 0
4      CHO("초"), // ordinal = 1
5      NONE("해당없음"); // ordinal = 2
6
7      public static final Set<Team> TEAMS = EnumSet.of(HAN, CHO);
8  }
```

TEAMS : elements= 011

CHO : 1 << e.ordinal() = 1 << 1 = 010

```
  011
& 100
-----
  000
```

ordinal() 존재 이유 : EnumSet

```
아직 존재하지 않으면 지정된 요소를이 세트에 추가합니다.

매개 변수 : e -이 세트에 추가 할 요소

보고:      통화 결과로 세트가 변경되면 true

던지기 :   NullPointerException - e 가 null 인 경우

public boolean add(E e) {  Complexity is 4 Everything is cool!
    typeCheck(e);

    long oldElements = elements;
    elements |= (1L << ((Enum<?>) e).ordinal());
    return elements != oldElements;
}
```

O(1) 시간복잡도로 추가 여부 조회

```
1  public enum Team {
2
3      HAN("한"), // ordinal = 0
4      CHO("초"), // ordinal = 1
5      NONE("해당없음"); // ordinal = 2
6
7      public static final Set<Team> TEAMS = EnumSet.of(HAN, CHO);
8  }
```

TEAMS : elements= 011

elements |= 1 << e.ordinal() = 100

```
  011
| 100
-----
  111
```

ordinal() 존재 이유 : EnumMap

EnumMap : Enum을 key로 사용하는 Map

```
public class EnumMap<K extends Enum<K>, V> extends AbstractMap<K, V> Comp
        implements java.io.Serializable, Cloneable
{
    이지도의 모든 키의 열거 유형에 대한 Class 객체.
    private final Class<K> keyType;

    K.를 구성하는 모든 값 (성능을 위해 캐시 됨)
    private transient K[] keyUniverse;

    이 맵의 배열 표현. ITH 요소는 우주 [i]가 현재 매핑 된 값 또는 아무것도 맵핑되지 않으면 널에 매핑되는 값
    private transient Object[] vals;

    이지도의 매핑 수.
    private transient int size = 0;
```

```
1  public enum Team {
2
3      HAN("한"),
4      CHO("초"),
5      NONE("해당없음");
6
7  }
```

keyUniverse = [HAN, CHO, NONE];

ordinal() 존재 이유 : EnumMap

EnumMap : Enum을 key로 사용하는 Map

```java
public class EnumMap<K extends Enum<K>, V> extends AbstractMap<K, V> Comp
        implements java.io.Serializable, Cloneable
{
    이지도의 모든 키의 열거 유형에 대한 Class 객체.
    private final Class<K> keyType;

    K.를 구성하는 모든 값 (성능을 위해 캐시 됨)
    private transient K[] keyUniverse;

    이 맵의 배열 표현. ITH 요소는 우주 [i]가 현재 매핑 된 값 또는 아무것도 맵핑되지 않으면 널에 매핑되는 값입
    private transient Object[] vals;

    이지도의 매핑 수.
    private transient int size = 0;
```

```java
public V put(K key, V value) {  Complexity is 4 Everything is cool!
    typeCheck(key);

    int index = key.ordinal();
    Object oldValue = vals[index];
    vals[index] = maskNull(value);
    if (oldValue == null)
        size++;
    return unmaskNull(oldValue);
}
```

```java
public enum Team {

    HAN("한"),
    CHO("초"),
    NONE("해당없음");

}
```

keyUniverse = [HAN, CHO, NONE];

vals = [73.5, 72.0, null]

O(1) 시간복잡도로 값에 접근 & 캐시 효율성

ordinal() 존재 이유 : EnumMap

EnumMap : Enum을 key로 사용하는 Map

```
public class EnumMap<K extends Enum<K>, V> extends AbstractMap<K, V>  Comp
        implements java.io.Serializable, Cloneable
{
    이지도의 모든 키의 열거 유형에 대한 Class 객체.
    private final Class<K> keyType;

    K.를 구성하는 모든 값 (성능을 위해 캐시 됨)
    private transient K[] keyUniverse;

    이 맵의 배열 표현. ITH 요소는 우주 [i]가 현재 매핑 된 값 또는 아무것도 맵핑되지 않으면 널에 매핑되는 값
    private transient Object[] vals;

    이지도의 매핑 수.
    private transient int size = 0;
```

내부적으로 배열을 사용하여 값을 저장

```
public V get(Object key) {  Complexity is 3 Everything is cool!
    return (isValidKey(key) ?
            unmaskNull(vals[((Enum<?>) key).ordinal()]) : null);
}
```

O(1) 시간복잡도로 값에 접근

ordinal() 존재 이유

O(1) 시간에 조회/삽입/삭제 가능

EnumMap, EnumSet 최적화

개요

1. ordinal() 존재 이유
2. **ordinal() 사용 단점**
3. ordinal() 대안
4. Summary

ordinal 단점

일반적인 애플리케이션 코드에서 ordinal()을 사용하는 것을 권장하지 않는다.

```
이 열거의 서수를 상수로 반환합니다 (초기 상수가 0의 서수가 할당 된 열거 선언에서의 위치). 대부분의 프로그래머는이 방법을 사용하지 않을 것입니다. java.util.EnumSet 및 java.util.EnumMap 과 같은 정교한 열거 기반 데이터 구조에서 사용하도록 설계되었습니다.

보고: 이 열거의 서수는 일정합니다

@Range(from = 0, to = java.lang.Integer.MAX_VALUE)
public final int ordinal() { return ordinal; }
```

ordinal 단점

1. 상수 선언 순서를 바꾸면 오동작한다.

```java
public enum Ensemble {
    SOLO, DUET, TRIO, QUARTET, QUINTET,
    SEXTET, SEPTE, OCTET, DOUBLE_QUARTET;

    public int numberOfMusicians() { return ordinal() + 1; }
}
```

```java
public enum Ensemble {
    SOLO, TRIO, DUET, QUARTET, QUINTET,
    SEXTET, SEPTE, OCTET, DOUBLE_QUARTET;

    public int numberOfMusicians() { return ordinal() + 1; }
}
```

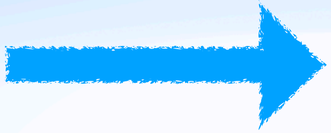
ordinal 단점

2. 이미 사용중인 정수와 값이 같은 상수는 추가할 수 없다.

```java
enum MusicalNote {
    C,      // ordinal = 0
    C_SHARP, // ordinal = 1
    D,      // ordinal = 2
    D_SHARP, // ordinal = 3
    E,      // ordinal = 4
    F,      // ordinal = 5
    F_SHARP, // ordinal = 6
    G,      // ordinal = 7
    G_SHARP, // ordinal = 8
    A,      // ordinal = 9
    A_SHARP, // ordinal = 10
    B;      // ordinal = 11

    // MIDI 노트 번호 계산 (C는 60, C#은 61, D는 62 등)
    public int getMidiNoteNumber() {
        return 60 + ordinal();
    }
}
```

C_SHARP = D_FLAT



```java
enum MusicalNote {
    C,      // ordinal = 0
    C_SHARP, // ordinal = 1
    // D_FLAT은 D_SHARP과 같은 MIDI 노트 번호를 가져야 함 (둘 다 61)
    D,      // ordinal = 2
    D_SHARP, // ordinal = 3
    E,      // ordinal = 4
    F,      // ordinal = 5
    F_SHARP, // ordinal = 6
    G,      // ordinal = 7
    G_SHARP, // ordinal = 8
    A,      // ordinal = 9
    A_SHARP, // ordinal = 10
    B;      // ordinal = 11

    // MIDI 노트 번호 계산 (C는 60, C#은 61, D는 62 등)
    public int getMidiNoteNumber() {
        return 60 + ordinal();
    }
}
```

ordinal 단점

3. 값을 중간에 비워두려면 쓰이지 않는 더미 상수를 추가해야한다.

```
enum MessageType {
    CONNECT,          // ordinal = 0
    DISCONNECT,       // ordinal = 1
    DATA,             // ordinal = 2
    RESERVED_3,       // ordinal = 3 (번호 유지를 위한 더미 상수)
    ACK,              // ordinal = 4
    ERROR;            // ordinal = 5

    public byte getTypeCode() {
        return (byte) ordinal();
    }
}
```

개요

1. ordinal() 존재 이유
2. ordinal() 사용 단점
3. **ordinal() 대안**
4. Summary

ordinal 대안

ordinal() 대신 인스턴스 필드에 저장하자

```
1  public enum Ensemble {
2
3      SOLO(1), DUET(2), TRIO(3), QUARTET(4), QUINTET(5),
4      SEXTET(6), SEPTET(7), OCTET(8), DOUBLE_QUARTET(8),
5      NONET(9), DECTET(10), TRIPLE_QUARTET(12);
6
7      private final int numberOfMusicians;
8
9      Ensemble(int size) { this.numberOfMusicians = size; }
10
11     public int numberOfMusicians() { return numberOfMusicians; }
12 }
```

개요

1. ordinal() 존재 이유
2. ordinal() 사용 단점
3. ordinal() 대안
4. Summary

Summary

1. ordinal() 존재 이유

- EnumSet 이나 EnumMap 같이 **열거 타입 기반의 범용 자료구조에 쓸 목적으로 설계**

- **O(1) 시간 복잡도로 조회/삽입/삭제 가능**

2. ordinal() 단점

1. 상수 선언 순서를 바꾸면 오동작한다.
2. 이미 사용중인 정수와 값이 같은 상수는 추가할 수 없다.
3. 값을 중간에 비워두려면 쓰이지 않는 더미 상수를 추가해야 한다.

3. ordinal() 대안

- ordinal() 대신 인스턴스 필드에 저장하자

꿋