

---

# 이펙티브 자바

item10 ~ item 11

24/01/16

---

---

# 3장 - 모든 객체의 공통 메서드

Object에서 final이 아닌 메서드(equals, hashCode, toString, clone, finalize)는 모두 재정의의  
염두에 두고 설계되었기에, 재정의 시 지켜야 할 일반 규약이 정의되어 있다.  
이 메서드들을 잘못 구현하면 대상 클래스가 일반적인 규약을 준수한다고 가정하고 만들어진 클래스  
(HashMap, HashSet 등)에서 오동작이 일어날 수 있다.

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## equals를 재정의하지 말아야 할 상황

1. 각 인스턴스가 본질적으로 고유하다.

➡ 값을 표현하는게 아니라 동작을 표현하는 클래스 (thread 등)

2. 인스턴스의 논리적 동치성을 검사할 일이 없다

➡ 예를들어 regex.Pattern(정규식)에서 equals를 재정의해서 정규표현식이 같은지 재정의할 필요 x

3. 상위 클래스에서 재정의한 equals가 하위 클래스에서도 같은 상황이다.

➡ 예를 들어 AbstractSet의 대부분의 Set 구현체는 equals를 그대로 사용

4. VO 여도 값이 같은 인스턴스가 2개 이상 안 만들어진다는 보장이 있다면 재정의안해도 된다.

➡ 예를들어 인스턴스 통제 클래스(싱글턴 등), Enum 등이 여기에 속한다.

어차피 논리적으로 같은 인스턴스가 2개 이상 만들어지지 x

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## equals를 재정의해야 할 상황

- '메모리주소를 기반으로 물리적으로 같은가?' 가 아니라 논리적 동치성(logical equality)를 비교해야할 때.
- 주로 값을 나타내는 Value Object 에 해당.
  - 즉, 객체가 같은지가 중요한게 아니라, 객체 내 값이 같은지 비교해야할 때.
- Map의 키, Set의 원소 등으로 사용하려면 재정의해야한다.
- equals 메소드를 재정의할 때는 동치관계를 구현해야 한다

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

```
<java />

public class StringEqualsExample {
    public static void main(String[] args) {
        String str1 = new String("Hello");
        String str2 = new String("Hello");
        String str3 = new String("World");

        // equals 메서드 호출
        System.out.println("str1.equals(str2): " + str1.equals(str2)); // true
        System.out.println("str1.equals(str3): " + str1.equals(str3)); // false
    }
}
```



String의 경우 equals는 메모리 주소가 아닌,  
값이 같은지를 판별한다.

이는 String 내부에서 equals를  
재정의하여 사용중이기 때문이다.

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## equals 메서드 재정의의 일반 규약: 동치관계

- 반사성(reflexivity)
  - : null이 아닌 모든 참조 값 x에 대해, `x.equals(x)`는 true다. (`x==x`)
- 대칭성(symmetry)
  - : null이 아닌 모든 참조 값 x, y에 대해, `x.equals(y)`가 true면 `y.equals(x)`도 true다. (`x==y`이면, `y==x`)
- 추이성(transitivity)
  - : null이 아닌 모든 참조 값 x, y, z에 대해, `x.equals(y)`가 true이고, `y.equals(z)`도 true면 `x.equals(z)`도 true다.
  - `x==y` , `y==z`이면, `x==z`
- 일관성(consistency)
  - : null이 아닌 모든 참조 값 x, y에 대해, `x.equals(y)`를 반복해서 호출하면 항상 true이거나 false다.
- null-아님
  - : null이 아닌 모든 참조 값 x에 대해, `x.equals(null)`은 false다.

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

## 대칭성

오른쪽 처럼 새롭게 정의한 String은, 일반 문자열과의 비교를 시도.

따라서

CaseInsensitiveString cis=new CaseInsensitiveString("hello");  
인스턴스가 있을 때,

String s="hello";와 비교를 하면 cis.equals(s)는 만족하나,  
s.equals(cis)는 만족하지 않는다.

String에는 CaseInsensitiveString에 대한 정의가 없기 때문이다.

따라서 String과 equals로 비교하려는 시도는 포기해야 한다.

```
<java />

public class CaseInsensitiveString {
    private String str;

    public CaseInsensitiveString(String str) {
        this.str = Objects.requireNonNull(str);
    }


    public boolean equals(Object o) {
        if (o instanceof CaseInsensitiveString) {
            return str.equalsIgnoreCase(((CaseInsensitiveString) o).str); //equalsIgnoreCase은 대소문자
        } else if (o instanceof String) {
            return str.equalsIgnoreCase((String) o);
        }
        return false;
    }
}
```

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## 추이성

상위 클래스에 없는 새로운 필드를 하위 클래스에 추가하고,  
새롭게 추가한 필드를 equals에서 무시한다면 이는 추이성을 위반하게 될 것이다.

 **NOTE** ex) x,y만 멤버변수로 갖던 클래스는 x,y를 비교하는 equals를 갖지만, 이를 z까지 확장한다면 z가 다른 경우는 걸러낼 수 없다.

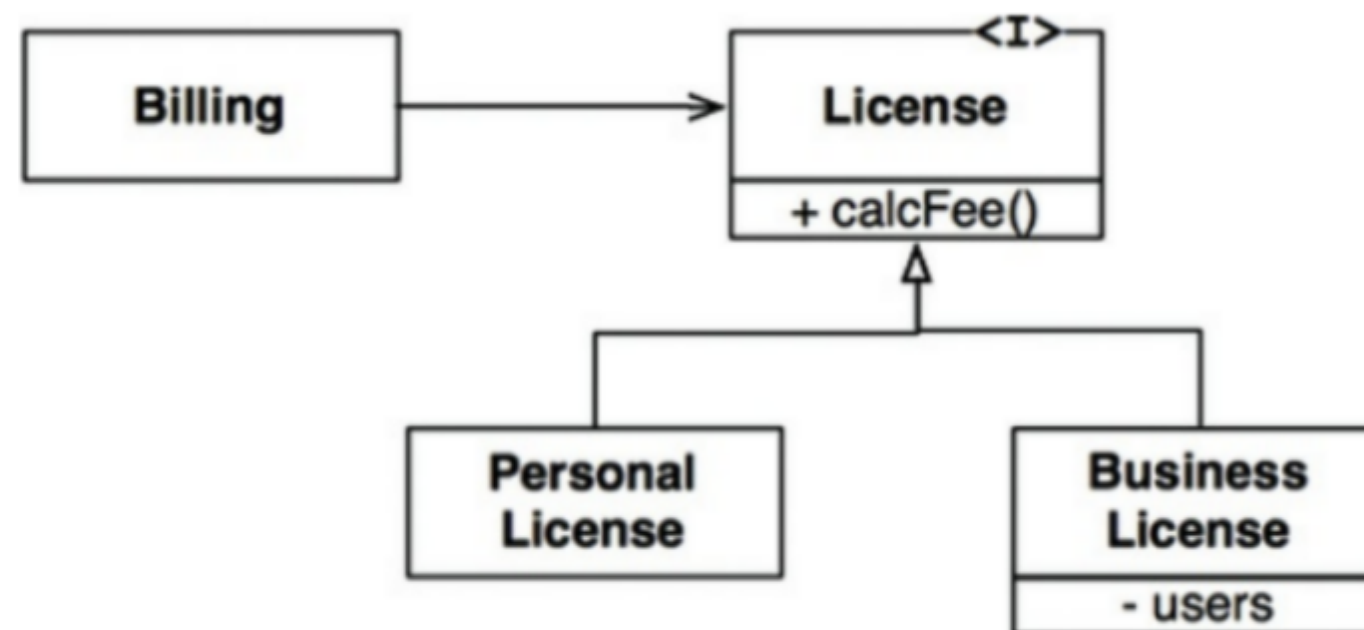
**리스코프 치환법칙에** 따르면 , 어떤 타입에 있어 중요한 속성이라면 그 하위타입에서도 똑같이 작동해야 한다.



# 아이템 10 equals는 일반 규약을 지켜 재정의하라

## LSP: 리스코프 치환 원칙 짚고 넘어가기

S 타입의 객체 o1 각각에 대응하는 T 타입 객체 o2가 있고,  
T 타입을 이용해서 정의한 모든 프로그램 P에서 o2의 자리에 o1을 치환하더라도  
P의 행위가 변하지 않는다면, S는 T의 하위 타입이다.



License 하위 타입 중 무엇을 사용하는지에  
전혀 의존하지 않기 때문에 , LSP 준수.

Figure 9.1 License, and its derivatives, conform to LSP

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

잘 정의된 인터페이스와 , 해당 인터페이스의 구현체끼리의 상호 치환 가능성이 곧 LSP.  
LSP를 어겼을 때 아키텍처에서는 어떤 일이 발생할까?

택시 파견 서비스

시스템이 고객에게 알맞은 기사를 선택하고 , 해당 기사 URI를 이용해 파견하는 서비스가 존재한다고 가정하자.

```
<code />
purplecab.com/driver/Bob
```

```
<code />
purplecab.com/driver/Bob
/pickupAddress/jayanglo
/pickupTime/153
/destination/ord
```

이때, 서로 다른 택시업체들이 destination 필드를 다른 방식으로 처리한다면 , (ex dest)

해당 업체들을 모듈 내에서 모두 조건문으로 다르게 처리해주어야 할 것이다.

효율도 떨어지고, 오류도 많이 발생하는 일이 될 것

---

여기서 택시업체들은, 한 인터페이스의 구현체들이라고 보면 된다.

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## 일관성

두 객체가 같다면, 수정되지 않는 한 영원히 같아야 하는 것.

---

## null아님

모든 객체가 null과 같지 않아야 한다.

x.equals(null) 은 false

null 에 대한 명시적 검사보단 instanceof 가 묵시적으로 검사해주므로, 이것만 사용하길 권장

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## 좋은 equals 재정의

1. == 연산자를 사용해 자기 자신의 참조인지 확인하라.
2. instanceof 연산자로 입력이 올바른 타입인지 확인하라. null검사도 해준다.
  - a. 서로 다른 클래스인 경우도 가아끔 있다. List, Map 등 컬렉션 인터페이스들의 구현체의 경우.
  - b. 이때는 equals가 해당 인터페이스를 사용해야 한다.
3. 입력을 올바른 타입으로 형변환하라. (2번을 했다면 100% 통과)
4. 입력 객체와 자기 자신의 대응되는 핵심 필드들이 모두 일치하는지 하나씩 검사하라. (하나라도 다르면 false)
5. 성능을 위해 다를 가능성이 더 크거나, 비용이 싼 필드를 먼저 비교해라.

# 아이템 10 equals는 일반 규약을 지켜 재정의하라

---

## 최고의 해결법

Java Lombok 라이브러리가 제공하는 @EqualsAndHashCode를 사용하자.

책에서는 AutoValue를 추천.

## 결론

웬만하면 equals를 재정의하지말자.

# 아이템 11 equals를 재정의하려거든 hashCode도 재정의하라

---

해시코드(hashCode)란 ? 해시 알고리즘에 의해 생성된 정수 값.

```
<java />

public class Test {
    public static void main(String[] args) {
        List<String> words = Arrays.asList("Gyunny", " Java", " Study");
        if (words.contains("Gyunny")) {
            System.out.println("Gyunny Java Love");
        }
    }
}
```

위 코드에서, contains의 시간복잡도는  $O(n)$ 이 되어야 하지만,

HashCode()를 사용해 키에 대한 해시 값을 계산하여 HashTable을 만든다면 데이터를 접근하는 시간이 비약적으로 빨라진다.

---

해시코드의 이점을 얻으려면, 하나의 버킷에 여러 원소가 쌓여 리스트 형태로 연결되는 것을 지양해야 한다.

# 아이템 11 equals를 재정의하려거든 hashCode도 재정의하라

---

## Object 명세에서 발췌한 hashCode 관련 규약.

- equals 비교에 사용되는 정보가 변경되지 않았다면, 애플리케이션이 실행되는 동안 그 객체의 hashCode 메서드는 몇 번을 호출해도 일관되게 항상 같은 값을 반환해야 한다.
  - 단, 애플리케이션을 다시 실행한다면 이 값이 달라져도 상관없다.
- equals(Object)가 두 객체를 같다고 판단했다면, 두 객체의 hashCode는 똑같은 값을 반환해야 한다.
- equals(Object)가 두 객체를 다르다고 판단했더라도, 두 객체의 hashCode가 서로 다른 값을 반환할 필요는 없다.
  - 단, 다른 객체에 대해서는 다른 값을 반환해야 해시테이블의 성능이 좋아진다.

따라서 equals를 재정의한 클래스 모두에서 hashCode도 재정의해야 한다.

# 아이템 11 equals를 재정의하려거든 hashCode도 재정의하라

---

좋은 hashCode 작성 방법 (실제 사용할만한 수준)

- 이미 정의된 hashCode(Integer.hashCode, Arrays.hashCode 등)를 최대한 사용해라
- 31을 곱해라. 홀수이면서 소수이기 때문이다.
- equals 비교에 사용되지 않은 필드는 hashCode 에서도 반드시 제외해야한다.

## 최고의 해결법

Java Lombok 라이브러리가 제공하는 @EqualsAndHashCode를 사용하자.