

equals는 일반 규약을 지켜 재정의하라

Effective Java Item 10

equals ?





equals

- 자바의 모든 클래스의 가장 최사위 클래스인 Object 클래스가 가진 메서드 중 하나
- 모든 클래스들은 equals 메서드를 사용할 수 있습니다.
- 두 인스턴스의 주소값을 비교하여 같은 인스턴스인지 확인하는 메서드

but 논리적으로 두 인스턴스가 동일하다고 판단될 때 equals 메서드를 사용하려면 equals 메서드를 재정의 해야 합니다.

equals



```
1 String string1 = new String("abcd");
2 String string2 = new String("abcd");
3
4 System.out.println(string1 == string2); // false, 메모리 주소 비교
5 System.out.println(string1.equals(string2)); // true, 내용 비교
```

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence,
        Constable, ConstantDesc {

    The value is used for character storage.
    Implementation This field is trusted by the VM, and is a subject to constant folding if String
    Note: is constant. Overwriting this field after construction will cause problems.
    Additionally, it is marked with Stable to trust the contents of the array. No
    facility in JDK provides this functionality (yet). Stable is safe here, because
    never null.

    @Stable
    private final byte[] value;
```

equalsIgnoreCase(String)

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    return (anObject instanceof String aString)
        && (!COMPACT_STRINGS || this.coder == aString.coder)
        && StringLatin1.equals(value, aString.value);
}
```

이미 String 클래스 안에는
equals 메서드가 재정의 되어있습니다.

equals 재정의하지 않아도 될때



equals 재정의하지 않아도 될때

- 1 각 인스턴스가 본질적으로 고유한 경우**
- 2 인스턴스의 논리적 동치성을 검사할 일이 없는 경우**
- 3 상위 클래스에서 재정의한 equals가 하위 클래스에도 딱 맞는 경우**
- 4 equals 메서드 호출할 일이 없는 경우**

1 각 인스턴스가 본질적으로 고유한 경우

값을 표현하기보다는 동작하는 개체를 표현하는 클래스는 자바와 같은 객체 지향 프로그래밍 언어에서 중요한 개념입니다.

이러한 클래스의 인스턴스들은 주로 고유한 동작이나 프로세스를 관리하며, 각 인스턴스는 본질적으로 다른 상태나 행위를 가집니다.

- Thread

이러한 클래스들은 모두 인스턴스가 고유한 동작이나 프로세스를 수행한다는 공통점을 가지고 있습니다.

- Runnable

- Socket

- HttpServlet

- TimerTask

이들은 대체로 상태를 관리하거나, 특정 작업을 수행하는데 초점을 맞추며, 그 과정에서 인스턴스 간의 고유성이 중요한 역할을 합니다. 따라서 이러한 유형의 클래스에서는 equals 메서드를 재정의하지 않고, object 클래스의 기본 구현을 사용하는 것이 일반적입니다.

equals를 재정의해야 할 때는 언제일까?





equals를 재정의해야 할 때

두 객체가 물리적으로 같은가가 아니라 논리적 동치성을 확인해야 하는데,
상위 클래스의 equals가 논리적 동치성을 비교하도록 재정의되지 않았을 때다.

주로 값을 표현하는 클래스인 값 클래스가 이런 경우



equals 메서드 재정의 시 일반 규약

equals
<pre>public boolean equals(Object obj)</pre>
Indicates whether some other object is "equal to" this one.
The equals method implements an equivalence relation on non-null object references: <ul style="list-style-type: none">• It is <i>reflexive</i>: for any non-null reference value <code>x</code>, <code>x.equals(x)</code> should return <code>true</code>.• It is <i>symmetric</i>: for any non-null reference values <code>x</code> and <code>y</code>, <code>x.equals(y)</code> should return <code>true</code> if and only if <code>y.equals(x)</code> returns <code>true</code>.• It is <i>transitive</i>: for any non-null reference values <code>x</code>, <code>y</code>, and <code>z</code>, if <code>x.equals(y)</code> returns <code>true</code> and <code>y.equals(z)</code> returns <code>true</code>, then <code>x.equals(z)</code> should return <code>true</code>.• It is <i>consistent</i>: for any non-null reference values <code>x</code> and <code>y</code>, multiple invocations of <code>x.equals(y)</code> consistently return <code>true</code> or consistently return <code>false</code>, provided no information used in equals comparisons on the objects is modified.• For any non-null reference value <code>x</code>, <code>x.equals(null)</code> should return <code>false</code>.
The equals method for class <code>Object</code> implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values <code>x</code> and <code>y</code> , this method returns <code>true</code> if and only if <code>x</code> and <code>y</code> refer to the same object (<code>x == y</code> has the value <code>true</code>).
Note that it is generally necessary to override the <code>hashCode</code> method whenever this method is overridden, so as to maintain the general contract for the <code>hashCode</code> method, which states that equal objects must have equal hash codes.

- **반사성 (reflexivity)** : null이 아닌 모든 참조 값 `x`에 대해 `x.equals(x)` 는 `true`이다.
- **대칭성 (symmetry)** : null이 아닌 모든 참조 값 `x`, `y`에 대해 `x.equals(y)`가 `true`이면 `y.equals(x)`도 `true`이다.
- **추이성 (transitivity)** : null이 아닌 모든 참조 값 `x`, `y`, `z`에 대해 `x.equals(y)`가 `true`이고 `y.equals(z)`도 `true`이면, `x.equals(z)`도 `true`이다.
- **일관성 (consistency)** : null이 아닌 모든 참조 값 `x`, `y`에 대해 `x.equals(y)`를 반복해서 호출하면 항상 `true`를 반환하거나 항상 `false`를 반환해야 한다.
- **null 아님** : null이 아닌 모든 참조 값 `x`에 대하여 `x.equals(null)`은 `false`이다.

1 반사성

null이 아닌 모든 참조 값 x 에 대해 $x.equals(x)$ 는 true이다.

객체는 자기 자신과 같아야 한다

2 대칭성

두 객체가 서로에 대한 동치 여부에 똑같이 답해야한다.

```
1 public final class CaseInsensitiveString {
2     private final String s; 3
3
4     public CaseInsensitiveString(String s) {
5         this.s = Objects.requireNonNull(s);
6     }
7
8     // 대칭성 부합x
9     @Override public boolean equals(Object o) {
10    1 if (o instanceof CaseInsensitiveString)
11        return s.equalsIgnoreCase(
12            ((CaseInsensitiveString) o).s);
13    if (o instanceof String) // 한 방향으로만 작동
14        return s.equalsIgnoreCase((String) o);
15    return false;
16 }
17
18 public static void main(String[] args) {
19     CaseInsensitiveString cis = new CaseInsensitiveString("Polish");
20     String s = "polish";
21
22    2 cis.equals(s); // true
23    s.equals(cis); // false
24 }
25 }
```

```
@Override public boolean equals(Object o) {
    return o instanceof CaseInsensitiveString &&
        ((CaseInsensitiveString) o).s.equalsIgnoreCase(s);
}
```

대칭성 명백히 위반

2 추이성

null이 아닌 모든 참조 값 x, y, z 에 대해
 $x.equals(y)$ 가 true이고 $y.equals(z)$ 도 true이면, $x.equals(z)$ 도 true이다.

A→B, B→C, A→C 이 관계

```
1 public class Point {
2     private final int x;
3     private final int y;
4
5     public Point(int x, int y) {
6         this.x = x;
7         this.y = y;
8     }
9
10    @Override public boolean equals(Object o) {
11        if(!o instanceof Point)
12            return false;
13        Point p = (Point) o;
14        return p.x == x && p.y == y;
15    }
16 }
```

```
1 public class ColorPoint extends Point {
2     private final Color color;
3
4     public ColorPoint(int x, int y, Color color) {
5         super(x, y);
6         this.color = color;
7     }
8
9     ...
10 }
```

1

대칭성 위배



```
1  @Override public boolean equals(Object o) {  
2      if(!o instanceof ColorPoint)  
3          return false;  
4      return super.equals(o) && ((ColorPoint) o).color == color;  
5  }
```



```
1  public static void main(){  
2      Point p = new Point(1,2);  
3      ColorPoint cp = new ColorPoint(1,2, Color.RED);  
4      p.equals(cp);    // true  
5      cp.equals(p);    // false  
6  }
```

ColorPoint의 equals는

입력 매개변수의 클래스 종류가 다르다며 매번 false만 반환

ColorPoint 타입 x 따라서 false

2 추이성 위배



```
1  @Override public boolean equals(Object o){
2      if(!(o instanceof Point))
3          return false;
4      if(!(o instanceof ColorPoint))
5          return o.equals(this);
6      return super.equals(o) && ((ColorPoint) o).color == color;
7  }
```



```
1  public static void main(){
2      ColorPoint p1 = new ColorPoint(1,2, Color.RED);
3      Point p2 = new Point(1,2);
4      ColorPoint p3 = new ColorPoint(1,2, Color.BLUE);
5      p1.equals(p2);    // true
6      p2.equals(p3);    // true
7      p1.equals(p3);    // false
8  }
```

대칭성은 지켜주지만 추이성을 깨버린다.

p1와 p2, p2와 p3 비교에서는 색상을 무시했지만,
p1과 p3비교에서는 색상까지 고려했기 때문!

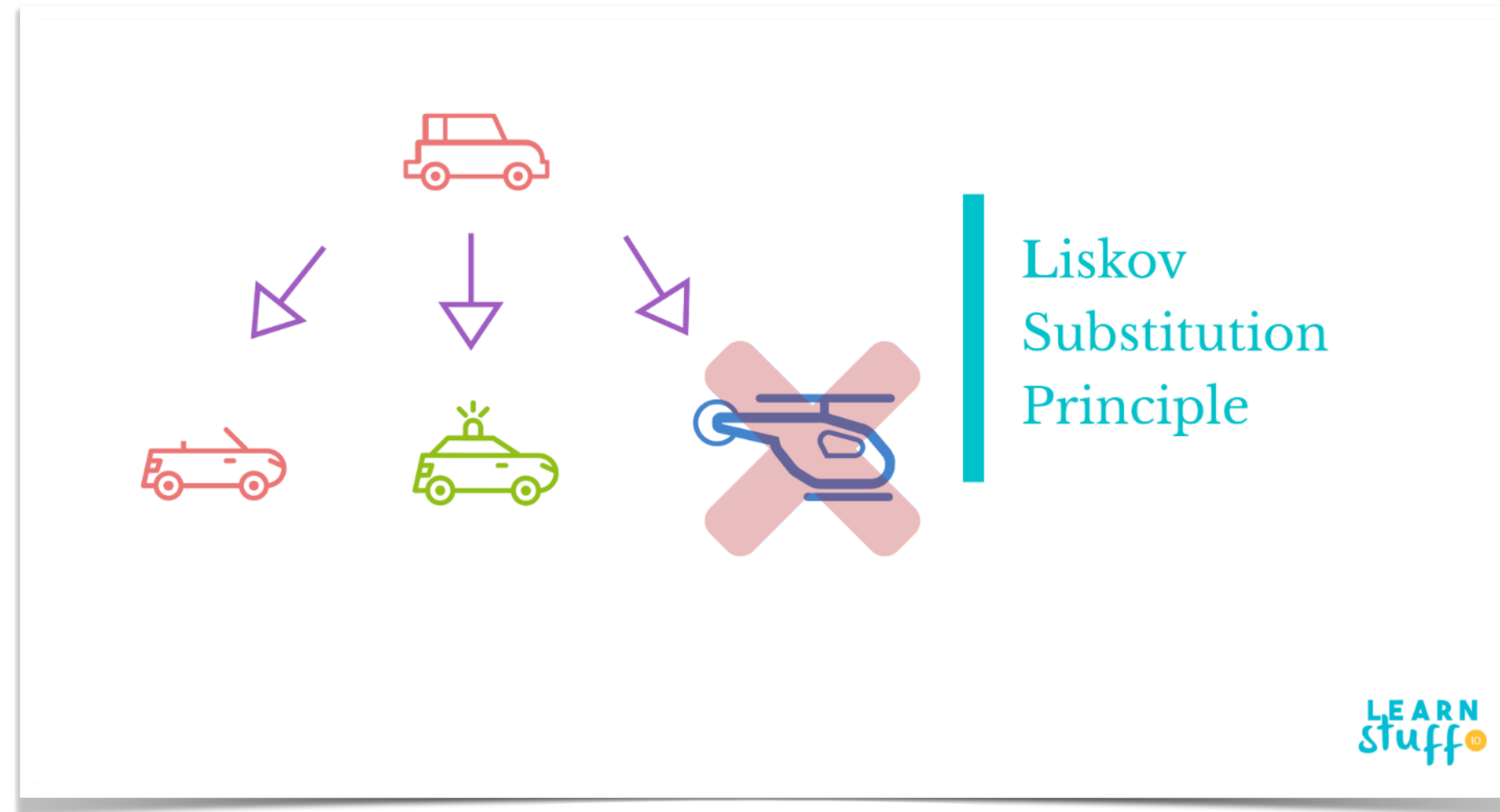
3 리스코프 치환 원칙

그럼 기존에 instanceof 대신 getClass를 사용하면 어떨까요?

```
1  @Override public boolean equals(Object o){  
2      if(o == null || o.getClass() != getClass())  
3          return false;  
4      Point p = (Point) o;  
5      return p.x == x && p.y == y;  
6  }
```

위의 코드는 같은 구현 클래스의 객체와 비교할 때만 true를 반환합니다

? 리스코프 치환 원칙



리스코프 치환 원칙은 1988년 바바라 리스코프(Barbara Liskov)가 올바른 상속 관계의 특징을 정의하기 위해 발표한 것으로, 서브 타입은 언제나 기반 타입으로 교체할 수 있어야 한다는 것을 뜻합니다.

3 리스코프 치환 원칙

그럼 기존에 instanceof 대신 getClass를 사용하면 어떨까요?

```
1  @Override public boolean equals(Object o){
2      if(o == null || o.getClass() != getClass())
3          return false;
4      Point p = (Point) o;
5      return p.x == x && p.y == y;
6  }
```

위의 코드는 같은 구현 클래스의 객체와 비교할 때만 true를 반환합니다

해결법 - 컴포지션 활용하기 (Not 상속)(Item 18)



```
1  public class ColorPoint{
2      private final Point point;
3      private final Color color;
4
5      public ColorPoint(int x, int y, Color color) {
6          point = new Point(x, y);
7          this.color = Objects.requireNonNull(color);
8      }
9
10     /* 이 ColorPoint의 Point 뷰를 반환한다. */
11     public Point asPoint(){ // view 메서드 패턴
12         return point;
13     }
14
15     @Override public boolean equals(Object o){
16         if(!(o instanceof ColorPoint)){
17             return false;
18         }
19         ColorPoint cp = (ColorPoint) o;
20         return cp.point.equals(point) && cp.color.equals(color);
21     }
22 }
```

4 일관성

일관성이란 null이 아닌 모든 참조 값 x , y 에 대해

`x.equals(y)`를 반복해서 호출하면 항상 `true`를 반환하거나 항상 `false`를 반환하는 것

```
while(x.equals(y) == x.equals(y))
```

5 null-아님

null이 아닌 모든 참조 값 x 에 대하여 $x.equals(null)$ 은 false이다.

즉, 모든 객체가 null과 같지 않아야 한다는 뜻이다.

잘못된 명시적 null 검사



```
1  @Override
2  public boolean equals(Object o) {
3      if(o == null) {
4          return false;
5      }
6  }
```

equals가 타입을 확인하지 않으면 잘못된 타입이 인수로 주어졌을 때 `ClassCastException`을 던져서 일반 규약을 위배하게 됩니다.

올바른 묵시적 null 검사



```
1  @Override
2  public boolean equals(Object o) {
3      if(!(o instanceof MyType)) {
4          return false;
5      }
6      MyType myType = (MyType) o;
7  }
```

instanceof는 첫 번째 피연산자가 null이면 false를 반환합니다.

따라서 입력이 null이면 타입 확인 단계에서 false를 반환하기 때문에 null 검사를 명시적으로 하지 않아도 됩니다.



양질의 equals 메서드 구현 방법

1. `==` 연산자를 사용해 입력이 자기 자신의 참조인지 확인한다.
2. `instanceof` 연산자로 입력이 올바른 타입인지 확인한다.
3. 입력을 올바른 타입으로 형변환한다.
4. 입력 객체와 자기 자신의 대응되는 '핵심' 필드들이 모두 일치하는지 하나씩 검사한다.



양질의 equals 메서드 구현 방법



```
1 public class Person {
2     private String name;
3     private int age;
4
5     public Person(String name, int age) {
6         this.name = name;
7         this.age = age;
8     }
9
10    @Override
11    public boolean equals(Object o) {
12        // == 연산자를 사용해 입력이 자기 자신의 참조인지 확인한다.
13        if (this == o) return true;
14
15        // instanceof 연산자로 입력이 올바른 타입인지 확인한다.
16        if (!(o instanceof Person)) return false;
17
18        // 입력을 올바른 타입으로 형변환 한다.
19        Person person = (Person) o;
20
21        // 입력 객체와 자기 자신의 대응되는 '핵심' 필드들이 모두 일치하는지 하나씩 검사한다.
22        return age == person.age &&
23            (name == person.name || (name != null && name.equals(person.name)));
24    }
25 }
```

세 가지만 자문

1 대칭적인가?

2 추이성이 있는가?

3 일관적인가?

⚠ equals 메서드 구현 주의 사항


1. equals를 재정의할 땐 hashCode도 반드시 재정의하자 (Item 11)
2. 너무 복잡하게 해결하려 들지 말자
 - 필드들의 동치성만 검사해도 equals 규약을 어렵지 않게 지킬 수 있다.
 - 오히려 너무 공격적으로 파고들다가 문제를 일으키기도 한다.
 - 일반적으로 별칭(alias)은 비교하지 않는 게 좋다.
3. Object 외의 타입을 매개변수로 받는 equals 메서드는 선언하지 말자

```
1  @Override public boolean equals(MyClass o){  
2      ...  
3  }
```

위 예제는 입력 타입이 Object가 아니므로 재정의가 아니라 다중정의한 것이됩니다.
equals 메서드의 매개변수는 반드시 Object 타입 이어야 합니다.



AutoValue

 google / auto

Q Type 🔍 to search | >_

Code

Issues 69

Pull requests 17


Discussions

Actions

Projects

Security

Insights

 auto Public

Watch 341

Fork 1.2k

Star 10.3k

main


15 Branches

94 Tags

Go to file t

Add file

<> Code

 dependabot[bot] and Google Java Core Libraries

Bump org.apache....


12acad0 · 9 hours ago


🕒 1,799 Commits


📁 .github	Bump actions/cache from 4.0.0 to 4.0.1	2 weeks ago
📁 common	Bump Truth to 1.4.2.	2 weeks ago
📁 factory	Bump org.apache.maven.plugins:maven-gpg-plugin from ...	9 hours ago
📁 service	Bump Truth to 1.4.2.	2 weeks ago
📁 util	Remove old release script.	2 years ago
📁 value	Bump kotlin.version from 1.9.22 to 1.9.23 in /value	4 days ago
📄 .gitignore	Ignore dependency-reduced pom spam	10 years ago
📄 CONTRIBUTING.md	Remove checkstyle references. It's unclear when the last ...	5 years ago
📄 LICENSE	Rename the LICENSE.txt file to LICENSE.	5 years ago
📄 README.md	Adjust documentation links to reflect the renamed main b...	2 years ago
📄 build-pom.xml	Delete Auto-Parent pom.xml entirely.	5 years ago


About


A collection of source code generators for Java.


 Readme


 Apache-2.0 license


 Code of conduct


 Security policy

 Activity

 Custom properties


 10.3k stars

 341 watching

 1.2k forks

Report repository

Releases 65

 AutoFactory 1.1.0

Latest

on Nov 21, 2023

+ 64 releases

마무리

꼭 필요한 경우가 아니면 equals를 재정의하지 말자, 많은 경우에 Object의 equals가

여러분이 원하는 비교를 정확히 수행해준다. 만약 재정의해야 할 때는 그 클래스의 핵심 필드 모두들 빠짐없이, 다섯 가지 규약을 확실히 지켜가며 비교해야 한다.



1 **but** 논리적으로 두 인스턴스가 동일하다고 판단될 때 equals 메서드를 사용하려면 equals 메서드를 재정의 해야 합니다.

2 일반 규약

반사성 : $a=a$

대칭성 : $a=b$, $b=a$

추이성 : $a=b$, $b=c$, $c=a$

일관성 : 어제 $a=b$ 오늘 $a=b$ 내일 $a=b$

null 아님 : $a.equals(null) = false$

3

🎯 양질의 equals 메서드 구현 방법

1. `==` 연산자를 사용해 입력이 자기 자신의 참조인지 확인한다.
2. `instanceof` 연산자로 입력이 올바른 타입인지 확인한다.
3. 입력을 올바른 타입으로 형변환한다.
4. 입력 객체와 자기 자신의 대응되는 '핵심' 필드들이 모두 일치하는지 하나씩 검사한다.

⚠ equals 메서드 구현 주의 사항

1. equals를 재정의할 땐 hashCode도 반드시 재정의하자 (Item 11)
2. Object 외의 타입을 매개변수로 받는 equals 메서드는 선언하지 말자

4 👍 AutoValue

5 꼭 필요한 경우가 아니면 equals를 재정의하지 말자