equals는 일반 규약을 지켜 재정의하라

최 혁

equals 메서드

```
    public boolean equals(Object obj) {
        return (this == obj);
    }
```

동일성(Identity): 두 객체의 참조값이 동일하다 (object identity)

동등성(Equality): 두 객체가 갖는 정보가 동일하다 (logical equality)

equals를 재정의하지 않는 것이 최선인 경우

1. **값을 표현하는 게 아닌 동작하는 개체를 표현하는 클래스의 경우**
   (각 인스턴스가 본질적으로 고유할 때)
2. **인스턴스의 logical equality를 검사할 일이 없을 때**
3. **상위 클래스에서 재정의한 equals가 하위 클래스에도 딱 들어맞을 때**
4. **클래스가 private이거나 package-private이고 equals 메서드를 호출할 일이 없을 때**

equals를 재정의할 경우

동등성을 확인해야 하는데, 상위 클래스의 equasl가 동등성을 비교하도록 재정의되지 않았을 때

- 주로 Integer, String같은 값 클래스
- Map, Set 사용 가능
- 객체가 싱글턴이 보장된다면 재정의하지 않아도 됨

equals 메서드 재정의 일반 규약

`equals 메서드는 equaivalence relation(동치관계)을 구현하며, 다음을 만족해야 한다`

- 반사성: 참조 값 x에 대해, x.equals(x)는 참이다
- 대칭성: 참조 값 x, y에 대해 x.equals(y)가 참이면 y.equals(x)도 참이다
- 추이성: 참조 값 x, y, z에 대해 x.equals(y)가 참이고 y.equals(z)가 참이면 x.equals(z)도 참이다
- 일관성: 참조 값 x, y에 대해 x.equals(y)를 반복해서 호출하면 항상 참이거나 거짓이어야 한다
- null 아님: 참조 값 x에 대해 x.equals(null)은 항상 거짓이다

반사성

```
null이 아닌 참조 값 x에 대해, x.equals(x)는 참이다
```

만약, 반사성을 어긴다면 contains 메서드가 원하는 대로 동작하지 않을 것이다.

대칭성

```
null이 아닌 참조 값 x, y에 대해 x.equals(y)가 참이면 y.equals(x)도 참이다
```

```java
public final class CaseInsensitiveString {

    private final String str;

    public CaseInsensitiveString(String str) {
        this.str = Objects.requireNonNull(str);
    }
    @Override
    public boolean equals(Object o) {
        if (o instanceof CaseInsensitiveString) {
            return str.equalsIgnoreCase(((CaseInsensitiveString) o).str);
        }

        if (o instanceof String) { // 한 방향으로만 작동한다.
            return str.equalsIgnoreCase((String) o);
        }
        return false;
    }
}
```

CaseInsensitiveString의 equals를 String과 연동하려고 하면 안 된다!

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

추이성

null이 아닌 참조 값 x, y, z에 대해 x.equals(y)가 참이고 y.equals(z)가 참이면 x.equals(z)도 참이다

```java
public class Point {

    private final int x;
    private final int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o) {
        if (!(o instanceof Point)) {
            return false;
        }
        Point p = (Point) o;
        return this.x == p.x && this.y == p.y;
    }
}
```

```java
public class ColorPoint extends Point {

    private final Color color;

    public ColorPoint(int x, int y, Color color) {
        super(x, y);
        this.color = color;
    }

//    대칭성 위반!
    @Override
    public boolean equals(Object o) {
        if(!(o instanceof ColorPoint))
            return false;
         return super.equals(o) && this.color == ((ColorPoint) o).color;
    }
}
```

```java
public class ColorPoint extends Point {

    private final Color color;

    public ColorPoint(int x, int y, Color color) {
        super(x, y);
        this.color = color;
    }

//    추이성 위반!
    @Override
    public boolean equals(Object o) {
        if(!(o instanceof Point))
            return false;
        if(!(o instanceof ColorPoint))
            return o.equals(this);
        return super.equals(o) && this.color == ((ColorPoint) o).color;
    }
}
```

```
ColorPoint p1 = new ColorPoint(1, 2, Color.RED);
Point p2 = new Point(1, 2);
ColorPoint p3 = new ColorPoint(1, 2, Color.BLUE);
p1.equals(p2);
p2.equals(p3);
p1.equals(p3);
```

추이성에 위배된다!

구체 클래스를 확장해 새로운 값을 추가하면서 equals 규약을 만족시킬 방법은 존재하지 않는다

(참고로 추상 클래스의 하위 클래스라면 equals 규약을 지키면서도 값을 추가할 수 있다. 상위 인스턴스를 생성할 수 없기 때문이다.)

구체 클래스의 하위 클래스에서 값을 추가할 방법은 없지만, 대신 우회 방법이 있다

```
public class ColorPoint {

private Point point;
private Color color;

public ColorPoint(int x, int y, Color color) {
    this.point = new Point(x, y);
    this.color = Objects.requireNonNull(color);
}

public Point asPoint() {
    return this.point;
}

@Override
public boolean equals(Object o) {
    if (!(o instanceof ColorPoint)) {
        return false;
    }
    ColorPoint cp = (ColorPoint) o;
    return this.point.equals(cp) && this.color.equals(cp.color);
}
```

일관성

`null이 아닌 참조 값 x, y에 대해 x.equals(y)를 반복해서 호출하면 항상 참이거나 거짓이어야 한다`

- **가변 객체는 비교 시점에 따라 서로 다를 수도 있고 같을 수도 있지만, 불변 객체는 한번 다르면 끝까지 달라야 한다**
- **클래스가 불변이든 가변이든 equals의 판단에 신뢰할 수 없는 자원이 끼어들게 해서는 안 된다** (URL / 매핑된 호스트의 IP주소)
  ㄴ 메모리에 존재하는 객체만을 사용한 계산만 수행해야 한다.

NULL 아님

```
@Override
public boolean equals(Object o) {
    // IntelliJ Default
    if (o == null || getClass() != o.getClass()) {
        return false;
    }

    // 저자 추천
    if (!(o instanceof Point)) {
        return false;
    }
}
```

equals 메서드 구현 방법 정리

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.

대칭성? 추이성? 일관성? -> 단위 테스트를 작성해보자!

귀찮으면 구글이 만든 AutoValue 프레임워크를 이용해보자!

핵심 정리

- 꼭 필요한 경우가 아니면 equals를 재정의하지 말자.
- 재정의를 해야 한다면 그 클래스의 핵심 필드를 모두 빠짐없이, 다섯 가지 규약을 확실히 지켜가며 비교하자

and

- 서로 다른 타입을 비교하려면 두 클래스 모두 equals에 로직을 수정하자
- 자식 클래스의 필드까지 동등성을 비교하려고 equals를 재정의하지 말자