Effective Java 읽기 스터디 20mins for weekly review

2주차 2021.03.01

Chapter 3

발표자: 김정수

목차

1. 3장 간단 리뷰

공통 메서드

Objects-> equals() 논리적 동치 여부

Objects-> hashCode() 해시 값 반환

Objects-> toString() 객체 정보 문자열 반환

Cloneable(I) -> clone() 복제된 객체 반환

Comparable(I) -> compareTo() 순서 반환

인터페이스나 상위 클래스에서 구현된 메서드는 하위 클래스에서 목적에 맞게 재정의하는 것이 권장된다.
단, 메서드별 규약을 지켜야 한다.

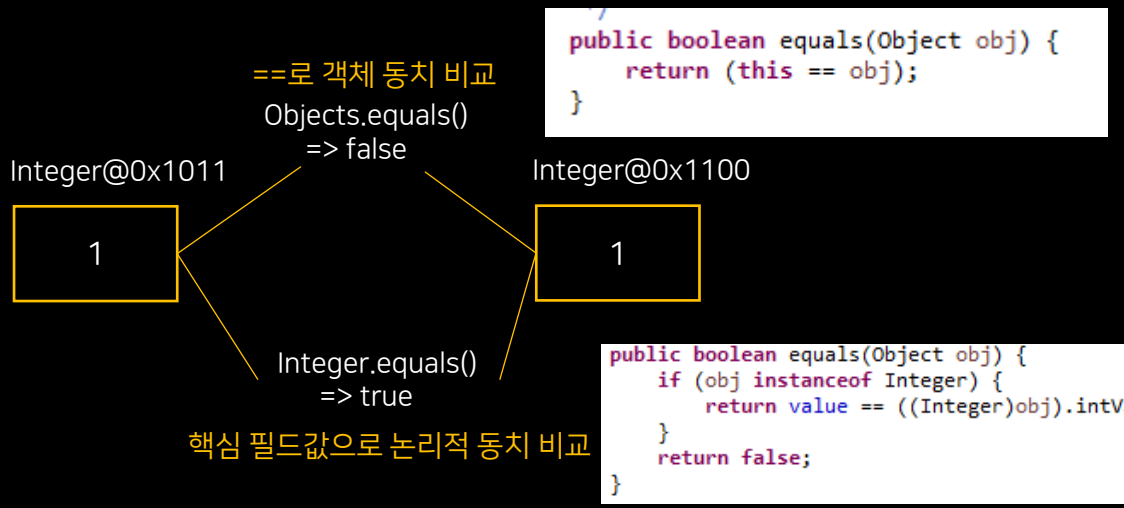
2. Item 10: equals() 재정의

재정의 하지 않아야 하는 상황

- 각 인스턴스가 고유해야 함
- 논리적 동치성 검사 필요 X
- 상속받은 걸로도 충분히 사용 가능한 경우
- private or package-private일 경우
- 싱글턴일 경우

재정의해야 하는 상황

- 상위 클래스의 equals로는 논리적 동치성을 검사할 수 없는 경우




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

```
public boolean equals(Object obj) {
    if (obj instanceof Integer) {
        return value == ((Integer)obj).intValue();
    }
    return false;
}
```

2. Item 10: equals() 재정의

equals 메서드 규약

- 반사성 : X=X

  x.equals(x) = true

- 대칭성 : X=Y -> Y=X

  If x.equals(y) == true, then y.equals(x) == true

- 추이성 : X=Y, Y=Z -> X=Z

  x.equals(y) == true and y.equals(z) == true, then x.equals(z) == true

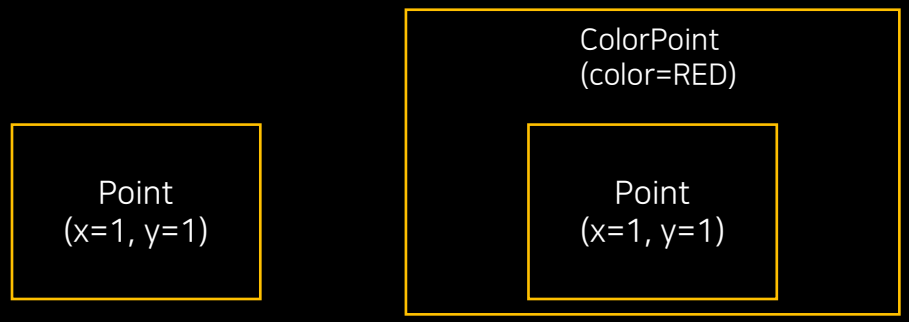
- 일관성 : 반복 호출 시 항상 같은 값 반환

  x.equals(y)==true라면 몇 번을 호출해도 true만 나옴

- null-아님 : x.equals(null)은 항상 false (x가 null이면 NullPointerException발생)

2. Item 10: equals() 재정의

구체 클래스를 확장하면서 규약을 만족시킬 수 없는 이유!

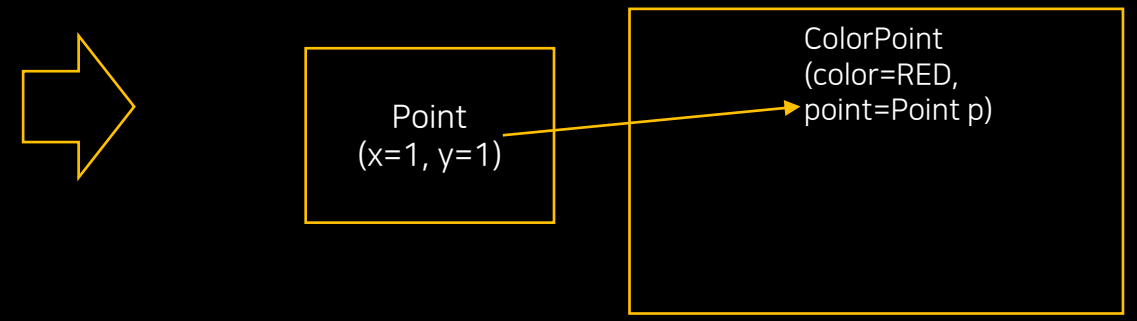


1) 대칭성을 위배한다.
A.equals(B) == true이지만 B.equals(A) == false다.
Point와 ColorPoint는 is-a관계이기 때문.

2) 추이성을 위배한다.
B.equals(A) == true를 만족시키려면 ColorPoint의 확장된 값 color를 무시한다.
color값만 다른 C가 있을 때, A=B이고, A=C이니 B=C가 성립되어버린다.

3) 리스코프 치환 원칙을 위배한다.
1)과 2)를 만족시키기 위해 ColorPoint끼리만 비교할 수 있게 할 경우 LSP를 깨뜨린다. LSP에 따르면 ColorPoint에서도 부모 클래스인 Point의 속성을 사용할 수 있어야 한다.

상속 대신 Composition으로 해결할 수 있다!



2. Item 10: equals() 재정의

```java
@Override
public boolean equals(Object o){
    if(o == this)
        return true;
    if(!(o instanceof MyClass))
        return false;
    MyClass obj = (MyClass)o;
    return obj.field1 == this.field1 && obj.field2 == this.field2 && ... ;
}
```

1. == 연산자를 사용해 입력이 올바른(자기 자신의) 참조인지 확인한다.
2. instanceof 연산자로 파라미터로 받은 객체가 올바른 타입인지 확인한다.
3. 올바른 타입으로 형변환한다.
4. 핵심 필드들이 모두 일치하는지 확인한다.

equals 재정의 Check List

대칭적인가?

추이성이 있는가?

일관적인가?

hashCode를 재정의했는가?

Object 타입을 매개변수로 받는가?

2. Item 11: hashCode() 재정의

hashCode

```
public int hashCode()
```

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer, provided no information used in `equals` comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.
- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

As much as is reasonably practical, the hashCode method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)

Returns:

a hash code value for this object.

x.equals(y)일때
x, y의 해시코드도 같아야 한다!

논리적 동치 비교에 쓰이는
핵심 필드로 해시코드를 만들자!

4. @AutoValue 프레임워크를 알아보자

```java
@AutoValue
abstract class Person {
    static Person create(String name, int age) {
        return new AutoValue_Person(name, age);
    }

    abstract String name();
    abstract int age();
}
```

AutoValue 프레임워크란?

⇒ 생성자, Getter, toString(), equals(), hashCode 자동으로 생성/재정의해주는 확장 프로그램이다.

⇒ abstract 클래스에 @AutoValue 어노테이션만 붙이면 클래스를 상속받는 모든 클래스에 자동으로 코드가 생성된다.

⇒ 리팩터링도 편리하게 해 준다.

⇒ 비슷한 프레임워크로 Lombok, Immutable이 있다.

4. @AutoValue 프레임워크를 알아보자

IDE로 value-type 클래스를 만들 때 자동 완성을 쓸 수는 있으나, 리팩토링에 손이 많이 간다.

```java
@AutoValue
public abstract class AutoValueMoney {
    public abstract String getCurrency();
    public abstract long getAmount();

    public static AutoValueMoney create(String currency, long amount) {
        return new AutoValue_AutoValueMoney(currency, amount);
    }
}
```

javac에서
코드를 재생성한다.

```java
public final class AutoValue_AutoValueMoney extends AutoValueMoney {
    private final String currency;
    private final long amount;

    AutoValue_AutoValueMoney(String currency, long amount) {
        if (currency == null) throw new NullPointerException(currency);
        this.currency = currency;
        this.amount = amount;
    }

    // standard getters

    @Override
    public int hashCode() {
        int h = 1;
        h *= 1000003;
        h ^= currency.hashCode();
        h *= 1000003;
        h ^= amount;
        return h;
    }

    @Override
    public boolean equals(Object o) {
        if (o == this) {
            return true;
        }
        if (o instanceof AutoValueMoney) {
            AutoValueMoney that = (AutoValueMoney) o;
            return (this.currency.equals(that.getCurrency()))
              && (this.amount == that.getAmount());
        }
        return false;
    }
}
```

abstract 클래스만으로도
안전한 equals 비교가 가능해진다.

변수명을 abstract 클래스에서 한 줄
바꾸면 value 클래스에 바로
리팩터링된다.
(이클립스는 리팩토링시키면 오래 걸림..)

References

- equals, hashCode 재정의(https://snoop-study.tistory.com/109)
- AutoValue (https://www.baeldung.com/introduction-to-autovalue)