이펙티브 자바 모든 객체의 공통 메서드

item10. equals는 일반 규약을 지켜 재정의하라

equals() 메서드는 객체의 내용이 동일한지 논리적 동치성을 확인하는 메서드이다.

equals를 구현하지 않아야 할 때

- 각 인스턴스가 본질적으로 고유할 때
- 인스턴스의 논리적 동치성을 검사할 일이 없을 때
- 상위 클래스에서 재정의한 equals()가 하위 클래스에서도 문제없이 이용 가능할 때

equals를 구현해여 할 때

- 상위 클래스에 equals() 를 재사용할 수 없고, 객체간의 논리적 동치성을 구현해야 할 때
- 주로 Integer, String과 같은 값 클래스의 경우.. 등등

equals() 메서드의 일반 규약

- null-아님 : null 이 아닌 모든 참조 값 x 에 대해 x.equals(null)은 false 이다.
- 반사성 : x.equals(x)는 true 이다.
- 대칭성 : x.equals(y)가 true이면, y.equals(x)도 true 이다.
- 추이성 : x.equals(y)가 true이고, y.equals(z)도 ture면, x.equals(z)도 ture 여야 한다.
- 일관성 : x.equals(y) 를 얼마나 반복하든, 결과는 항상 같아야 한다.

자바 API에서 equals()를 이용하는 내용은 모든 클래스가 위 규약을 지킨다고 가정함.

equals() 메서드를 구현하는 절차

- == 연산자로 자신의 참조인지 먼저 확인한다.
- instanceof 연산자로 입력이 올바른 타입인지 확인한다. - null 방지 및 캐스팅 에러 방지
- 입력을 올바른 타입으로 형변환
- 기본 타입은 == 비교, float, double은 Float.compare()와 같은 정적 메서드 활용(부동소수점)
- 배열의 모든 원소가 핵심 필드 → Arrays.equals() 활용
- null 값을 정상 값으로 취급 → Object.equals(Object, Object)메서드 방지 가능

equals() 메서드 구현 주의 사항

- equals()를 재정의할 땐 반드시 hashCode()도 재정의한다.
- Object 외의 타입을 파라미터로 받는 equals()를 만들지 마라
- IDE에서 제공하는 기능을 사용하자

item11. equals를 재정의하려거든 hashCode도 재정의하라

hashCode일반 규약

- equals()를 재정의한 클래스 모두에서 hashCode()도 재정의해야 한다 → 안하면 HashMap, HashSet에서 문제가 발생함
- equals()비교에 사용되는 필드가 변하지 않았다면, hashCode() 메서드는 몇번을 호출하든, 항상 같은 값을 반환해야 한다.
  - 단, 애플리케이션을 재시작한 경우에는 달라질 수 있다.
- equals(Object)가 두 값을 같다고 판단했다면, hashCode() 의 반환 값도 같아야 한다.

- equals(Object)가 두 객체를 다르다고 판단했더라도, hashCode() 가 달라질 필요는 없다.
    - 단, 해시테이블 성능 최적화를 위해서 다르게 나오는 것이 좋다.

hashCode()를 모두 동일한 값을 반환하도록 설정한다면?

hashMap이 동일한 key인지 판단할 때, equals()와 hashCode() 두 메서드 모두에 의존하기 때문이다. 하나라도 다르면, 다른 key로 간주한다.

hashCode()에서 계속 똑같은 숫자를 반환해도 동작은 하지만, 원소가 늘어날수록 해시 테이블의 성능이 매우 떨어진다. O(1)의 시간복잡도가 점차 링크드리스트처럼 O(n)이 된다.

좋은 해시함수는 32비트 정수 범위에 인스턴스들을 고루 분배해야 한다.

hashCode() 메서드를 구현하는 요령

핵심필드란 equals() 비교에 사용되는 필드이다.

- 핵심 필드에 아래 작업을 수행한다.
    - 기본 타입 필드면, Type.hashCode(f)를 수행한다.
        - Type이란, 기본 타입의 박싱 클래스를 말한다.
    - 참조 타입 필드면서, 이 클래스의 equals() 메서드가 이 필드의 equals() 메서드를 재귀적으로 호출한다면, 필드의 hashCode()를 재귀적으로 호출하면 된다. (보통 equals()가 있다는 건 hashCode()도 올바른 방식으로 구현했음을 말한다.)
        - 계산이 더 복잡해질 것 같으면, 표준형을 만들어 표준형의 hashCode()를 호출한다.
        - 필드의 값이 null이면 0을 사용한다.
    - 배열이라면, 핵심원소 각각을 별도 필드처럼 다룬다.
        - 배열에 핵심원소가 하나도 없다면 0(권장) 혹은 다른 상수를 사용한다.
        - 모든 원소가 핵심 원소라면 Arrays.hashCode()를 사용한다.
- 위의 작업에서 계산한 해시코드로 result를 갱신한다.

    - 첫 필드 값이라면 할당
        - int result = c
    - 첫 필드 값이 아니라면 갱신
        - result = 31 * result + c
- result를 반환한다.

사실 hashCode()를 직접 만드는 것보다 IDE의 도움을 받자.

item12. toString을 항상 재정의하라.

toString을 재정의하면 좋은점

- 디버깅이 쉬워진다. → 의미있는 정보를 볼 수 있음
- Map과 같은 경우, 내부에 많은 값을 가지는데, toString을 재정의하면 모든 값을 알 수 있어 좋다.

toString을 재정의시 주의점

- 객체가 가진 주요 정보는 모두 보여주는 것이 좋다.
- 정보가 너무 많다면 간단하게 요약된 정보를 보여줘도 된다.

toString을 재정의하면 이점이 많기에 Object의 toString()은 모든 구체 클래스에서 재정의하면 좋다.

item13. clone 재정의는 주의해서 진행하라

```java
package java.lang;


/**
 * 클래스는 Cloneable 인터페이스를 구현하여 해당 메서드가 해당 클래스 인스턴스의 필드 간 복사본을 만드는 것이 적법함을
 * Object.clone() 메서드에 나타냅니다.
 *
 * Cloneable 인터페이스를 구현하지 않는 인스턴스에서 Object의 복제 메소드를 호출하면
 * CloneNotSupportedException 예외가 발생합니다.
 *
 * 규칙에 따라 이 인터페이스를 구현하는 클래스는 공용 메서드를 사용하여 Object.clone (보호됨)을 재정의해야 합니다. 이 메
 * 서드 재정의에 대한 자세한 내용은 Object.clone() 참조하세요.
 *
 * 이 인터페이스에는 clone 메소드가 포함되어 있지 *않습니다* . 따라서 단순히 이 인터페이스를 구현한다는 사실만으로 객체를 복
 * 제하는 것은 불가능합니다. clone 메소드가 반사적으로 호출되더라도 성공한다는 보장은 없습니다.
 *
 * 부터:          JDK1.0
 * 또한보십시오: CloneNotSupportedException ,
 *              Object.clone()
 * 작가:          등록되지 않은
 */
public interface Cloneable {
}
```

cloneable의 역할

- 복제해도 되는 클래스임을 나타내는 믹스인 인터페이스이다.
- cloneable 인터페이스는 clone() 메서드의 동작 방식을 결정한다.
- Cloneable을 구현하지 않은 인스턴스에서 clone()을 호출하면 CloneNotSupportedException을 던진다.

Cloneable의 문제점

- 일반적인 인터페이스의 동작방식과 다르게 상위 Object 클래스에 protected 접근자로 된 clone() 메서드가 존재하고, 그걸 오버라이드 해야 한다. (믹스인으로 의도해서 만들었는데, 믹스인이라고 말하기 뭔가 애매하다.)
- Cloneable만 사용하면 당연히 복제가 이뤄질 줄 알았는데 생각보다 복잡한 구조를 이해하고 있어야 한다.
- 자바의 기본 의도와 다르게 생성자를 호출하지 않고 객체를 생성할 수 있게 되어버린다.

Clone() 메서드의 일반 규약

- x.clone() != x 식은 참이어야 한다.
    - 복사된 객체가 원본이랑 같은 주소를 가지면 안된다는 뜻이다.

- x.clone().getClass() == x.clone().getClass() 식도 참이어야 한다.
  - 복사된 객체가 같은 클래스여야 한다는 뜻이다.
- x.clone().equals()는 참이어야 하지만, 필수는 아니다.
  - 복사된 객체가 논리적 동치는 일치해야 한다는 뜻이다. (필수는 아니다.)

- 인터페이스를 만들 때는 절대 Cloneable을 확장해선 안된다.
  - Cloneable은 클래스의 믹스인(사용) 의도로 만들어진 것이다.
- final 클래스라면 Cloneable을 구현해도 위험은 크지 않지만, 성능 최적화 관점에서 검토 후에 드물게 허용해야 한다.
- **복제 기능은 생성자와 팩터리를 이용하는 것이 최고이다.**
  - **단 한가지 예외는 배열을 복사할 때이다.**

item14. Comparable을 구현할지 고려하라

Comparable은 믹스인 인터페이스이며, compareTo에 같은 객체끼리의 natrual order를 정의한다.

Comparable 구현의 이점

- Comparable을 구현한 객체의 배열은 쉽게 정렬 가능하다.
- Collection 객체들에서도 정렬을 활용할 수 있다.
  - TreeSet 자료구조 같은 경우, Comparable을 구현한 타입만 제너릭으로 받을 수 있다.
    - String 타입을 넣는 경우, 들어간 모든 문자열을 알파벳순으로 출력 가능하다.

compareTo 메서드의 일반 규약

주어진 객체를 기준으로하여 아래와 같은 값을 반환한다.

- 비교대상보다 작으면 **음의 정수(-1)**
- 비교대상과 같으면 **0**
- 비교대상보다 크면 **양의 정수(1)**

sgn은 부호 함수(signum function)를 의미하며, 음수, 0, 양수일 때 각각 -1, 0, 1로 표현하도록 하였다.

- Comparable을 구현한 클래스는
  - 모든 x, y에 대해 sgn(x.compareTo(y)) == -sgn(y.compareTo(x))여야 한다.
    - 예외도 x.compareTo(y)와 y.compareTo(x)가 동일하게 터져야 한다.
  - 추이성(transitivity)을 보장해야 한다. (x.compareTo(y) > 0 && y.compareTo(z) > 0)이면, x.compareTo(z) > 0이여야 한다.
    - 0보다 크다는 것은 비교 대상보다 크다는 것이다. x > y > z 인 경우에 x > z여야 한다는 뜻이다.
  - x.compareTo(y) == 0이면, sgn(x.compareTo(z)) == sgn(y.compareTo(z))여야 한다.
    - x == y일 때, x == z && y == z여야 한다는 뜻이다.
  - (x.compareTo(y) == 0) == (x.equals(y)) 는 꼭 지켜야하는 것은 아니지만 권고사항이다.
    - equals()와 논리적 동치를 판단하는 기준이 같다는 뜻이다.
    - 이를 지키지 않으면, 컬렉션 인터페이스(Collection, Set, Map)에서 정의된 동작과 엇박자를 낼 수 있다. 정렬된 컬렉션은 동치를 비교할 때 equals()대신 compareTo()를 사용한다.

정리하자면 반사성, 대칭성, 추이성을 지켜야 한다는 뜻이다.

equals()와 달리 타입이 다른 객체에 대해서는 신경 안 써도 된다.

equals()와 같이 상속으로는 이러한 일반규약을 다 지킬 방법이 없고, '사용'형태로 객체 안에 사용할 필드를 두는 것이 낫다.

- 순서가 있는 클래스를 작성한다면, Comparable 인터페이스를 구현하는 것이 좋다.
- compareTo 메서드를 구현할 때는 박싱 클래스에서 제공하는 compare()를 적극 활용하자.