The Hidden Contract beetween Equals and Compareable

# Contract breaking

We know we use equals for check two objects are meaningfully equal or not, on other hand by comparable interface we compare two objects. But Has there is any relation between them?

Or we have to maintain contract between them?

We know When we override equals we have to override hashcode as they maintain a contract.

But we can implement comparable or may not. If we don’t implement comparable then we can’t compare objects in other words we can’t do sorting on them.

So enable sorting, either we introduce Comparable or Comparator( if the class is a legacy class , third-party class or needs to sort in a different way).

So in both case of Comparable and Comparator, returns int, which says

Positive: Current instance is greater than Other.

Zero: Equals

Negative: Current instance is smaller than Other.

So if Comparable returns 0 it means two objects are same by comparison. Also By equals, we check two objects are same if it returns true.

So my question is If Comparable shows two objects are same and equals says both are un-equals then we face any problem?

Let’s, examine it by code,

**package** com.example.contract;

**public** **class** Glass **implements** Comparable<Glass>{

**public** **enum** Size{

***BIG***(3),***MEDIUM***(2),***SMALL***(1);

**private** **int** size;

Size(**int** size)

{

**this**.size=size;

}

**public** **int** getSize()

{

**return** size;

}

};

**private** String material;

**private** Size size;

**public** Glass(Size size,String material)

{

**this**.size=size;

**this**.material=material;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result

+ ((material == **null**) ? 0 : material.hashCode());

result = prime \* result + ((size == **null**) ? 0 : size.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Glass other = (Glass) obj;

**if** (material == **null**) {

**if** (other.material != **null**)

**return** **false**;

} **else** **if** (!material.equals(other.material))

**return** **false**;

**if** (size != other.size)

**return** **false**;

**return** **true**;

}

@Override

**public** **int** compareTo(Glass o) {

**if**(**this**.size.equals(o.size))

{

**return** 0;

}

**else** **if**(**this**.size.getSize() > o.size.getSize())

{

**return** 1;

}

**else**

{

**return** -1;

}

}

@Override

**public** String toString() {

**return** "Glass [material=" + material + ", size=" + size + "]";

}

}

**package** com.example.contract;

**import** java.util.HashSet;

**import** java.util.Set;

**import** java.util.TreeSet;

**public** **class** ContractBreaker {

**public** **static** **void** main(String[] args) {

Glass plastic = **new** Glass(Glass.Size.***BIG***,"Plastic");

Glass glass = **new** Glass(Glass.Size.***BIG***,"glass");

Set<Glass> set = **new** HashSet<Glass>();

set.add(plastic);

set.add(glass);

System.***out***.println(set);

Set<Glass> treeSet = **new** TreeSet<Glass>();

treeSet.add(plastic);

treeSet.add(glass);

System.***out***.println(treeSet);

}

}

Output :

HashSet says [Glass [material=Plastic, size=BIG], Glass [material=glass, size=BIG]]

TreeSet says [Glass [material=Plastic, size=BIG]]

Oops, While HashSets treats them, unequal, TreeSet treats them equal

So in programming, if we change HashSet to TreeSet or vice versa, weird results shows.

**The problem occurs because,**

When we implement equals we consider two properties of Glass i.e. material and size, so if both are equals then we say both Glasses are meaningfully equivalent.

But in CompareTo method we only consider size. If two size is same, we treat them equals.

Here we did the wrong thing,

HashMap,ArrayList,HashSet ,they add element based on equals method so When we use HashMap it treats two objects as different objects as their material is different.

But TreeMap ,TreeSet use Compare method so TreeSet treats them as the same Object.

By this Program, we see that there is a hidden Contract between equals and comparable.

***Contract:***

***If two objects are equals by equality check then those object must be equaled by Comparable or Comparator test.***

**In Java BigDecimal breaks this contract so when use BigDecimal, please remember the scenario stated above.**