Eliminating a Problem: Why Programmers don't Need to Implement Equality

Beate Ritterbach

University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
beate.ritterbach@studium.uni-hamburg.de

1 Problem

Implementing equality in object-oriented languages (e.g. via the method `equals` in Java) holds many pitfalls and doing it wrongly may result in inconsistent behavior. This problem has been discussed in various articles, blogs and text books. Here are some of the most significant problems in that area:

- It is hard to ensure that equality adheres to the "contract", and it is especially difficult to program equality as an equivalence relation, i.e. reflexive, symmetric and transitive ([1]).
- Languages like Java, C#, Scala, etc. use a hashcode to manage collection elements. That's why equal objects must have the same hashcode. Yet, programmers easily fail to implement the method that calculates the hashcode corresponding to the method that checks equality.
- Even if equality adheres to the contract and the hashcode-rule is satisfied, equality may cause strange effects. If it depends on mutable state it may result in contradictory behavior, e. g. when you add an element to a collection, change it and later try to retrieve it ([3]).

It is amazing that a (seemingly) simple and fundamental concept like equality is so hard to implement. Thinking about it, it is even more peculiar *that* equality has to be implemented by the programmer at all. One might expect it to be a basic function of every programming language.

2 Relevance

Equality is at the basis of many other operations. A correct implementation of equality is practically relevant in nearly every kind of application. Surveys have shown that many Java-implementations of the `equals`-method (both in practical projects and in text books) are flawed with respect to the contract ([2], [4]). This is even more crucial because this kind of error is a "silent killer": It will cause neither compile time errors nor run time exceptions, but it will result in unexpected behavior in some different place, making it hard to track down the actual cause. If there were an approach that makes it possible to provide equality as a built-in language feature, one substantial source of error would be eliminated.

3 Research Contribution

For the problem of implementing equality correctly many solutions have been proposed. Each solution describes an elaborate implementation technique, and most solutions primarily focus on compliance to the contract. The techniques vary in some details (e.g., referring to solutions in Java, whether it is preferable to use `instanceof` or `getClass` for checking the type of the operands). However, all solutions share a certain complexity, and they involve a large amount of programming conventions. Our contribution pursues a radically different approach.

- It claims that there are two meanings of equality: *similarity* and *value equality* (and that this is the cause of many of the problems outlined above).
- It takes into account the distinction between *objects* and *values*, which aids in distinguishing the two meanings of equality more clearly.
- This, in turn, enables the programming language to *generate* (value) equality, thus turning it from a programmer-defined method into a built-in language feature.

This approach can ease implementing and using equality in many respects:

- First and foremost, implementation by the programmer - which often is cumbersome and easily leads to errors - is no longer required.
- Additionally, with this approach the language can prevent inadvertent tests of incomparable types (see also [4]).
- Equality can also involve null values without causing null pointer exceptions (or their equivalents), thus making equality more symmetric.
- The hashcode-rule (s.a.) is always satisfied automatically.
- There is no way of using a "wrong" comparison, similar to, e.g., using the operator `==` to check for equal strings in Java.

The language-generated equality does adhere to the contract (even to contracts that are more strict than the one described by Bloch ([1]), e. g. a contract that prohibits dependance on mutable state, like in [4]). However, since equality is a built-in operation, the rules of the contract fade from the spotlight of the programmer's attention since she does no longer need to take care of them.

In short: instead of solving the problem of how to implement equality, the proposed approach eliminates that necessity.

References

1. Bloch, J.: Effective Java (2nd Edition). Addison-Wesley, Amsterdam (2008)
2. Langer, A., Kreft, K.: Secrets of equals() Part 1: Not all implementations of equals() are equal. Java Solutions April 2002 (2002)
3. Odersky, M., Spoon, L., Venners, B.: How to Write an Equality Method in Java. `http://www.artima.com/lejava/articles/equality.html`
4. Vaziri, M., Tip, F., Fink, S., Dolby, J.: Declarative Object Identity Using Relation Types. In: ECOOP 2007, Vol. 4609, pp. 54–78. Springer (2007)