# Order-independent Hash Algorithm

Ask Question

I am currently working on a collection library for my custom programming language. I already have several data types (Collection, List, Map, Set) and implementations for them (mutable and immutable), but what I was missing so far was `hashCode` and `equals`. While these are no problem for Lists as they are ordered collections, the play a special role for Sets and Maps. Two Sets are considered equal if they have the same size and the same elements, and the order in which the Sets maintain them should not make a difference in their equality. Because of the equals-hashCode-contract, the `hashCode` implementation also has to reflect this behavior, meaning that two sets with the same elements but different ordering should have the same hash code. (The same applies for Maps, which are technically a Set of Key-Value-Pairs)

**Example** (Pseudocode):

```
let set1: Set<String> = [ "a", "b", "c" ]
let set2: Set<String> = [ "b", "c", "a" ]
set1 == set2       // should return true
set1.hashCode == set2.hashCode // should also return true
```

How would I implement a reasonably good hash algorithm for which the `hashCode`s in the above example return the same value?

java algorithm hash set

edited Dec 13 '17 at 10:46

Cœur

**14k** 6 87 119

asked Jun 9 '15 at 14:23

Clashsoft

**5,431** 4 26 50

How about a pair (sum,product) of the terms in the set? Both of them together would not be common for different sets of numbers (as far as I have seen). – Anindya Dutta Jun 9 '15 at 14:28

For example something like `(e1.hashCode() + e2.hashCode() + ... + en.hashCode()) ^ (e1.hashCode() * e2.hashCode() * ... * en.hashCode())` ? – Clashsoft Jun 9 '15 at 14:30

1 Did you try to have a look at how Java implements this? – RealSkeptic Jun 9 '15 at 14:31

Just did, it sums the `hashCode` of the elements – Clashsoft Jun 9 '15 at 14:32

2 There is a big difference between "hash of the sum" and "sum of the hashes". The former, as your examples indicate, is problematic. The latter has fewer problems provided the individual hashes are well-distributed over a large range. – rici Jun 9 '15 at 16:18

## 3 Answers

The JDK itself proposes the following solution to this problem. The contract of the java.util.Set interface states:

> Returns the hash code value for this set. The hash code of a set is defined to be the sum of the hash codes of the elements in the set, where the hash code of a null element is defined to be zero. This ensures that s1.equals(s2) implies that s1.hashCode()==s2.hashCode() for any two sets s1 and s2, as required by the general contract of Object.hashCode().

An alternative to using the sum of the entries' hash codes would be to use, for example, the `^` (XOR) operator.

The Scala language uses an ordering-invariant version of the Murmurhash algorithm (cf. the private `scala.util.hashing.MurmurHash3` class) to implement the `hashCode` (or `##`) method of its immutable sets and similar collections.

edited Jun 9 '15 at 15:07

answered Jun 9 '15 at 14:44

Dirk
**25.7k** 5 64 89

As I stated in the comments, I already found the JDK solution for this problem, but I want to know about more useful unordered collection hash algorithms with less collision potential. – Clashsoft Jun 9 '15 at 14:49

@Clashsoft What collision potential? If just one of the individual hash codes works well, the entire hash algorithm will be evenly distributed. – btilly Jun 9 '15 at 15:15

@btilly The distribution of a sum of uniform random variables is not uniform! – augurar Jul 13 '17 at 23:39

@augurar There are very important differences between real numbers and 32-bit signed integers. This is one of them. The people writing `java.set.Util` knew what they were doing and came up with a good strategy here. – btilly Jul 14 '17 at 6:32

You can calculate the hash sum sorting your collection in alphabetical order.

There is the C# sample - I hope you can translate it in Java :)

```
static String GetHash(List<String> l)
{
    using (System.Security.Cryptography.MD5 md5 =
System.Security.Cryptography.MD5.Create())
    {
        return BitConverter.ToString(md5.ComputeHash(l.OrderBy(p =>
p).SelectMany(s => System.Text.Encoding.ASCII.GetBytes(s +
(char)0)).ToArray())).Replace("-", "");
    }
}
```

answered Oct 24 '17 at 11:15

Dimitri Sorokin

**16** 2

---

Here's the pseudocode for a possible implementation:

```
String hashCode = null;
for(element : elements){
    hashCode = xor(hashCode, getHashCode(element));
}
return hashCode;
```

The `xor` function should return a string that is as long as the longest of the two arguments. It will XOR the bits in each until it gets to the end of one of the arguments. It will then take the remaining bits from the longer string and append those on.

This implementation will mean that the hashCode of a set will be as long as the hashCode of its longest element. Because you are XORing the bits, at the end the hashcode will be the same regardless of the

order of your elements. However, as with any hashing implementation, there will be the chance for collisions.

answered Jun 9 '15 at 14:43

Briguy37

**6,921** 1 21 44

But what would I do with a `String` when I need an `int` hashCode? This seems like a very resourceful solution. – Clashsoft Jun 9 '15 at 14:48

@Clashsoft I wasn't sure if you wanted an `int` or a `String`. If it's just an int, then taking the sum of the hashCodes of the individual elements will get you what you need, as long as overflows wrap around instead of causing errors. If overflows cause errors, then you'll need to handle that case explicitly and wrap manually. Same concept though. – Briguy37 Jun 9 '15 at 14:50

Thanks for the answer, but I want to find a different solution other than summing the hash codes of the elements (see comments). – Clashsoft Jun 9 '15 at 14:52

@Clashsoft - Note that it really depends on your implementation of hashing numbers whether [1,4] will collide with [2,3]. Remember, you are summing the hash of the number and not the number. Also, though the string implementation is more resource intensive, that also means it will have more bits in the hash and thus less of a chance for collisions. – Briguy37 Jun 9 '15 at 14:58

Typically, the hash code of an integer (if you are not going super-security) is the integer itself. So `1.hashCode == 1`. Also, since `hashCode` is enforced to be an `int`, I have to use `string.hashCode` in the end anyway. – Clashsoft Jun 9 '15 at 15:03