two unequal objects with same hashcode

Hashcode() and equals() concept is

> 1) If two Objects are equal according to equal(), then calling the hashcode method on each of those two objects should produce same hashcode.

and other one is

> 2) It is not required that if two objects are unequal according to the equal(), then calling the hashcode method on each of the two objects must produce distinct values.

I tried and understood first one and this is the code for first point.

```
public class Test
{
public static void main(String[] args) {

Map<Integer, Integer> map=new HashMap<Integer, Integer>();
map.put(1, 11);
map.put(4, 11);
System.out.println(map.hashCode());
Map<Integer, Integer> map1=new HashMap<Integer, Integer>();
map1.put(1, 11);
map1.put(4, 11);
System.out.println(map1.hashCode());
if (map.equals(map1)) {
    System.out.println("equal ");
}
}
}
```

the above program gives same hashcode for two different objects.

Can someone explain me with an example,how can two different objects which are unequal according to the equals() have same hashcode.

java hashmap equals hashcode

asked May 6 '13 at 14:17
sahu
46 2 9

5 Compare the number of possible hash codes to the number of possible `Long` s or `String` s. en.wikipedia.org/wiki/Pigeonhole_principle – SLaks May 6 '13 at 14:19

possible duplicate of Example of ==, equals and hashcode in java – cHao May 6 '13 at 14:21

6 Answers

> 2) It is **not required** that if two objects are **unequal** according to the equal(), then calling the hashcode method on each of the two objects must produce distinct values.

Depending on the hashing function, 2 different objects can have the same hash code. However, 2 objects wich are the same must produce the same result when hashed (unless someone implemented a hashing function with random numbers in which case it's useless)

For example, if I am hashing integers and my hashing function is simply `(n % 10)` then the number `17` and the number `27` will produce the same result. This does not mean that those numbers are the same.

answered May 6 '13 at 14:20
Jonathan

506 1 4 20

hashCode() has 32-bit possible values. Your objects can have much more than this so you are going to have some objects with the same hashCode, i.e. you cannot ensure they will be unique.

This is made worse in a hash collection of a limited size. The maximum capacity of HashMap is 1 << 30 or about one billion. This means that only 30 bits are really used and if your collection doesn't use 16+ GB and is only say one thousand buckets (or 1 << 10 technically) then really you have only 1000 possible buckets.

Note: on the HotSpot JVM, the default Object.hashCode() is never negative i.e. only 31-bit, though I am not sure why.

If you want to generate lots of objects with the same hashCode look at Long.

```
// from Long
public int hashCode() {
    return (int)(value ^ (value >>> 32));
}

for(long i = Integer.MIN_VALUE; i < Integer.MAX_VALUE;i++) {
    Long l = (i << 32) + i;
    System.out.print(l.hashCode()+" ");
    if (i % 100 == 0)
        System.out.println();
}
```

This will generate 4 billion Long all with a hashCode of 0.

edited May 6 '13 at 14:27 answered May 6 '13 at 14:20
Peter Lawrey
252k 24 245 477

1 If you want to generate lots of objects with the same hashCode, would not it be simpler to override "public int hashChode()" to return the same hashCode. It is not final. – emory May 6 '13 at 14:48

Example with Strings (all the strings below have a hashcode of 0):

```
public static void main(String[] args) {
    List<String> list = Arrays.asList("pollinating sandboxes",
                                      "amusement & hemophilias",
                                      "schoolworks = perversive",
                                      "electrolysissweeteners.net",
                                      "constitutionalunstableness.net",
                                      "grinnerslaphappier.org",
                                      "BLEACHINGFEMININELY.NET",
                                      "WWW.BUMRACEGOERS.ORG",
                                      "WWW.RACCOONPRUDENTIALS.NET",
                                      "Microcomputers: the unredeemed lollipop...",
                                      "Incentively, my dear, I don't tessellate a
derangement.",
                                      "A person who never yodelled an apology,
never preened vocalizing transsexuals.");
    for (String s : list) {
        System.out.println(s.hashCode());
    }
}
```

(stolen from this post).

answered May 6 '13 at 14:22
assylias
115k 10 173 308

+1 I was thinking of this post ;) – Peter Lawrey May 6 '13 at 14:22

It's pretty simple actually,

First we have to know what a hash code is.

In java, a hash code is simple a 32 bit signed integer that is somehow derived from the data in question. The integer types are usually just (Int Data) Mod (some reasonable large prime number).

Let's do a simple hash on integers.
Define:

```
public int hash(int num){ return num % 19 ; }
```

In this case, both 19 and 38 will return the hash value of 0.

For string types, the hash is derived from the individual characters and each ones position in the string, divided by a reasonably large number. (Or, in the case of Java, ignoring overflow in a 32 bit sum).

Given that there are arbitrarily many strings possible, and there is a limited number of hashcodes (2^32) for a string, the pigeon-hole principle states that there are at least two different strings that result in the same hashcode.

answered May 6 '13 at 14:29
Chris Cudmore
12.2k 6 38 78

The purpose of `hashCode` is to enable the following axiom and corollary:

- If one happens to know the hash codes of two objects, and those hash codes don't match, one need not bother examining the objects any further to know that the objects won't match. Even if two arbitrarily-chosen non-matching objects would have a 10% chance of having matching hash codes, testing hash codes would let one eliminate 90% of the comparisons one would otherwise need. Not as big a win as eliminating 99.99%, but definitely worthwhile nonetheless.

- Knowledge that none of the objects in a bunch have a particular hash code implies that none of the objects in that bunch will match an object with that hash code. If one partitioned a collection of objects into those whose hash code was an even number and those whose hash was odd, and one wanted to find whether one had a given item whose hash code happened to be even, there would be no need to examine anything in the collection of of odd-hash items. Likewise there would be no need to look for an odd-hash item in the even-hash collection. Even a two-value hash could thus speed up searches by almost half. If one divides a collection into smaller partitions, one can speed things up even further.

Note that `hashCode()` will offer the most benefit if every different item returns a different hash, but it can offer substantial benefit even when many items have the same hash value. The difference between a 90% savings and a 99.99% savings is often much greater than the numbers would suggest, and thus one if one can reasonably easily improve things to 99%, 99.9%, or better one should do so, but he difference between having zero false matches and having a few false matches in a collection is pretty slight.

answered Dec 6 '13 at 18:06
supercat
29.1k 1 38 66

My understanding is that hashCode is a numeric representation of the memory address, but is not the actual address. It can be changed, without affecting the actual address. Thus, it should be possible to set all objects to the same hashCode, even if they are all entirely different things. Think of everybody on one block all suddenly having the same street address. They are truly different people, but now all share the same street address. Their house didn't move, a mischevious teen just labeled everybody as "100 N. Main".

I am pretty new to Java, so take my reply with a bit of caution.

answered May 6 '13 at 14:23
petematt62
1 1

This is actually a completely incorrect idea about what a hashcode is. – Chris Cudmore May 6 '13 at 14:32

What is a better, or the correct, idea? – petematt62 May 6 '13 at 14:36