how is hashCode() implemented in Java

Ask Question

How is hashCode() implemented?

My assumption is that it uses the object memory location as the initial number (the seed) on which it runs the hash function. However, this is not the case.

I've also looked at Hash : How does it work internally? but it does not answer my question.

Yes I could download the SDK, but before I do that and look at the code, perhaps someone else already has knowledge of it.

Thanks :)

EDIT: I know it should be overridden and such, so please try to stay on topic :)

java hash

edited May 23 '17 at 11:46
Community ♦
1 1

asked Apr 11 '12 at 13:23
Adrian
3,770 6 35 67

1 From your own link: *"This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java(TM) programming language."* Of course this is at the discretion of the implementation, but that should confirm your assumptions about the Oracle implementation. Why do you say it's not the case? – Mark Peters Apr 11 '12 at 13:25

It is having own native implementation if you see Object class - public native int hashCode(); – Subhrajyoti Majumder Apr 11 '12 at 13:45

@MarkPeters and what happens when the objects are moved in memory? – Adrian Apr 11 '12 at 13:59

5 Answers

Of course it is implementation specific, but generally the hash code for an object will be computed lazily and stored in the object header. Odd things are done with headers to keep them small whilst allowing complex locking algorithms.

In the OpenJDK/Oracle JVM the usual method of computing the initial hash code is based on the memory address at the time of the first request. Objects move about in memory, so using the address each time would not be a good choice. The hash code isn't the actual address - that would typically be a multiple of eight which isn't great for using straight in a hash table particularly with a power of two size. Note identity hash codes are not unique.

HotSpot has build time options to always use zero or use a secure random number generator (SRNG) for testing purposes.

answered Apr 11 '12 at 13:38
Tom Hawtin - tackline
122k 25 177 264

the multiple of 8 might be multiplied by a prime number to give a better hashable number – Adrian Apr 11 '12 at 13:54

@Adrian It'd still be a multiple of eight. Really it gets shifted about. – Tom Hawtin - tackline Apr 11 '12 at 17:40

2 you're right, I dont know what I was thinking – Adrian Apr 12 '12 at 13:24

1 You are only partially correct. **True** that it is implementation specific. But **false** that usual method is based on the memory address. This is a very common mistake - I'm not blaming you because even javadoc from the `Object#hashCode()` is misleading. – G. Demecki Nov 17 '14 at 15:25

@GrzesiekD. I believe it is historically correct. – Tom Hawtin - tackline Nov 17 '14 at 17:22

No, no, no. All answers in this thread are wrong or at least only partially correct.

First: `Object.hashCode()` is a native method, so its implementation **depends solely** on the JVM. It may vary between HotSpot and other VM implementations like JRockit or IBM J9.

If you are asking:

> how is `hashCode()` implemented in Java?

Then the answer is: it depends on which VM you are using.

Assuming that you are using Oracle's default JVM—which is HotSpot, then I can tell you that HotSpot has six `hashCode()` implementations. You can choose it using the `-XX:hashCode=n` flag running JVM via command line, where `n` can be:

```
0 – Park-Miller RNG (default)
1 – f(address, global_statement)
2 – constant 1
3 – Serial counter
4 – Object address
5 – Thread-local Xorshift
```

The above is copied from this post.

And if you dig a little bit around in the HotSpot source code, you may find below snippet:

```
if (hashCode == 0) {
  value = os::random();
} else {
  ...
```

`os::random()` is just the implementation of the Park-Miller Pseudo Random Generator algorithm.

That's all. **There isn't any notion of memory address.** Although two other implementations, `1` and `4`, use an object's memory address, the default one doesn't use it.
The notion that `Object.hashCode()` is based on the object's address is largely a historic artefact - it is no longer true.

I know that inside `Object#hashCode()` JavaDoc we can read:

> (...) this is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.

But it is obsolete and misleading.

edited Nov 15 '17 at 11:52
Just a student
5,340 1 19 49

answered Nov 17 '14 at 15:21
G. Demecki
6,168 1 34 46

The implementation of the hashcode() function varies from Object to Object. If you want to know how a specific class implements hashcode(), you'll have to look it up for that class.

answered Apr 11 '12 at 13:39
IrAndroid
2,429 12 26

The hashCode method defined by class Object returns distinct integers for distinct objects. This could be implemented by converting the internal address of the object into an integer (but this implementation style is not required by the standard). It gets interesting with new classes which override hashCode in order to support hash tables (equal and hashCode): http://www.javapractices.com/topic/TopicAction.do?Id=28

answered Apr 11 '12 at 13:43
hburde
1,318 9 6

I'm assuming you're talking about the `Object` implementation of `hashCode` , since the method can and should be overridden.

It's implementation dependent. For the Sun JDK, it's based on the object's memory address.

answered Apr 11 '12 at 13:26
Taymon
14.8k 6 43 71

1 I think that's somewhat misleading. Objects are moved around in memory but have a stable identity hash code. – Tom Hawtin - tackline Apr 11 '12 at 13:32

@TomHawtin-tackline exactly why I asked this question; it seems like a good seed (say multiplied by a prime), but after you realize objects are moved in memory... IDK – Adrian Apr 11 '12 at 13:56

Downvote due to two reasons. Firstly: usually there is no need to override `hashCode` . Secondly: Default for Sun JDK is pseudo random number, not object's memory address. You can see my post to find out a little. – G. Demecki Nov 17 '14 at 15:28

@Adrian hash code is stable (although being random) because after being computed the first time, the result is then stored in the object's header and is returned on subsequent calls to hashCode(). – G. Demecki Nov 17 '14 at 15:31