HowToDoInJava

Java hashCode() and equals() Methods

Last Updated: January 30, 2022 | By: Lokesh Gupta | Java Basics | Java Equals, Java Hashcode

Learn about Java `hashCode()` and `equals()` methods, their **default implementation, and how to correctly override them**. Also, we will learn to implement these methods using 3rd party classes `HashCodeBuilder` and `EqualsBuilder`.

> **`hashCode()`** and **`equals()`** methods have been defined in `Object` class which is parent class for all java classes. For this reason, all java objects inherit a default implementation of these methods.

Table of Contents:

1. Uses of hashCode() and equals() Methods

1. `equals(Object otherObject)` – verifies the equality of two objects. Its default implementation simply checks the object references of two objects to verify

their equality.
By default, two objects are equal if and only if they are refer to the same memory location. Most Java classes override this method to provide their own comparison logic.

2. `hashcode()` – returns a unique integer value for the object in runtime.
By default, integer value is derived from memory address of the object in heap (but it's not mandatory).
The object's hash code is used for determining the index location, when this object needs to be stored in some HashTable like data structure.

1.1. Contract between hashCode() and equals()

Overriding the the `hashCode()` is generally necessary whenever `equals()` is overridden to maintain the general contract for the `hashCode()` method, which states that **equal objects must have equal hash codes**.

- Whenever it is invoked on the same object more than once during an execution of a Java application, the **`hashCode()` must consistently return the same integer**, provided no information used in `equals` comparisons on the object is modified.
This integer need not remain consistent between the two executions of the same application or program.
- **If two objects are equal** according to the `equals()` method, then calling the `hashCode()` on each of the **two objects must produce the same integer** result.
- It is ***not* required that if two objects are unequal** according to the `equals()`, then calling the `hashCode()` on each of the **both objects must produce distinct integer** results.
However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

2. Overriding the Default Behavior

Everything works fine until we do not override any of both methods in our classes. But, sometimes, the application needs to change the default behavior of some objects.

Let us understand **why we need to override equals and hashcode** methods.

2.1. The default behavior of Employee class

Let's take an example where your application has `Employee` object. Let us create a minimal possible structure of `Employee` class:

```
public class Employee
{
    private Integer id;
    private String firstname;
    private String lastName;
    private String department;

    //Setters and Getters
}
```

Above `Employee` class has some fundamental attributes and their accessor methods. Now consider a simple situation where you need to compare two Employee objects. Both employee objects have the same `id`.

```
public class EqualsTest {
    public static void main(String[] args) {
        Employee e1 = new Employee();
        Employee e2 = new Employee();

        e1.setId(100);
        e2.setId(100);

        System.out.println(e1.equals(e2));  //false
    }
}
```

No prize for guessing. The above method will print "*false*."

But is it correct after knowing that both objects represent the same employee? In a real-time application, this should return `true`.

2.2. Should we override only equals() method?

To achieve correct application behavior, we need to override `equals()` method as below:

```
public boolean equals(Object o) {
    if(o == null)
    {
        return false;
    }
    if (o == this)
    {
        return true;
    }
    if (getClass() != o.getClass())
    {
        return false;
    }

    Employee e = (Employee) o;
    return (this.getId() == e.getId());
}
```

Add this method to the `Employee` class, and `EqualsTest` will start returning `"true"`.

So are we done? Not yet. Let's test the above-modified `Employee` class again in a different way.

```
import java.util.HashSet;
import java.util.Set;
```

```
public class EqualsTest
{
    public static void main(String[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee();

        e1.setId(100);
        e2.setId(100);

        //Prints 'true'
        System.out.println(e1.equals(e2));

        Set<Employee> employees = new HashSet<Employee>();
        employees.add(e1);
        employees.add(e2);

        System.out.println(employees);  //Prints two objects
    }
}
```

The above example prints two objects in the second print statement.

If both employee objects have been equal, in a `Set` which stores unique objects, there must be only one instance inside `HashSet` because both objects refer to the same employee. What is it we are missing??

2.3. Overriding hashCode() is necessary

We are missing the second important method `hashCode()`. As java docs say, if we override `equals()` then we ***must*** override `hashCode()`. So let's add another method in our `Employee` class.

```
@Override
public int hashCode()
{
    final int PRIME = 31;
```

```
        int result = 1;
        result = PRIME * result + getId();
        return result;
    }
```

Once the above method is added in Employee class, the second statement starts printing only a single object in the second statement and **thus validating the true equality of e1 and e2**.

3. EqualsBuilder and HashCodeBuilder

Apache commons provide two excellent utility classes HashCodeBuilder and EqualsBuilder for generating hash code and equals methods.

We can use these classes in the following manner.

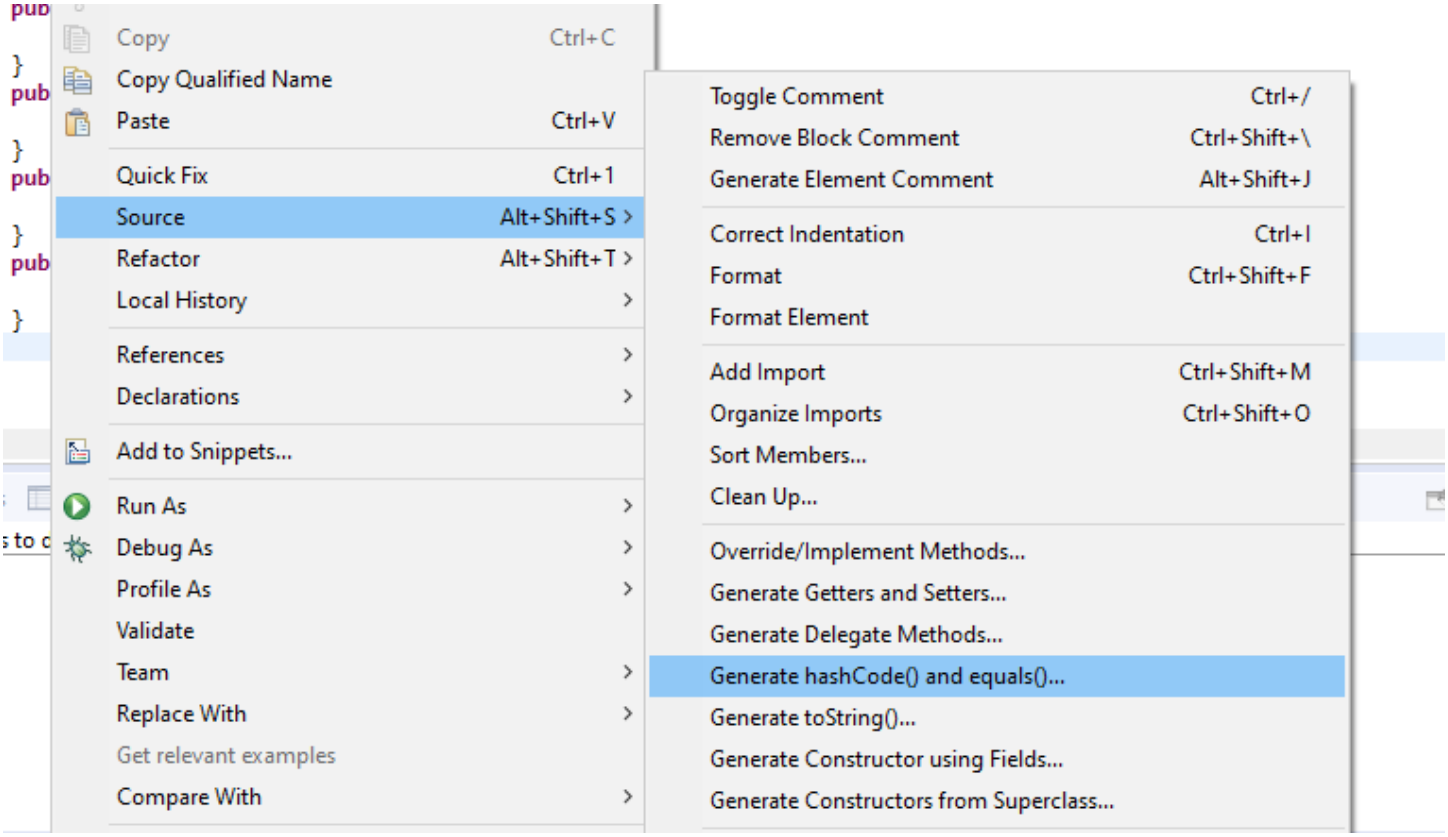
```
import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;
public class Employee
{
    private Integer id;
    private String firstname;
    private String lastName;
    private String department;

    //Setters and Getters

    @Override
    public int hashCode()
    {
        final int PRIME = 31;
        return new HashCodeBuilder(getId()%2==0?getId()+1:getId(), PR
    }

    @Override
    public boolean equals(Object o) {
    if (o == null)
```

```
        return false;

    if (o == this)
        return true;

    if (o.getClass() != getClass())
        return false;

    Employee e = (Employee) o;

    return new EqualsBuilder().
              append(getId(), e.getId()).
              isEquals();
    }
}
```

4. Generating hashCode() and equals() in Eclipse IDE

Most editors provide common source code templates. For example, **Eclipse IDE** has an option to generate an excellent implementation of `hashCode()` and `equals()`.

> Right click on Java file -> Source -> Generate hashCode() and equals() ...



Generate hashCode() and equals() In Eclipse

5. Best Practices

1. Always use the same fields to generate `hashCode()` and `equals()`. As in our case, we have used employee `id`.
2. The `equals()` must be *consistent* (if the objects are not modified, then it must keep returning the same value).
3. Whenever **a.equals(b)**, then *a.hashCode()* must be same as *b.hashCode()*.
4. If we override one method, then we should override the other method as well.

6. Special Attention When Using in *ORM*

If you're dealing with an ORM, make sure always to use getters **and never use the field references in `hashCode()` and `equals()`**. Because in ORM, occasionally fields are lazy loaded and not available until we call their getter methods.

For example, In our `Employee` class if we use `e1.id == e2.id`. It is very much possible that `id` field is lazy-loaded. So, in this case, id field inside the methods might be zero or `null`, and thus resulting in incorrect behavior.

But if uses `e1.getId() == e2.getId()`, we can be sure even if the field is lazy-loaded, calling the field getter will populate the field first.

If you feel I am missing something or wrong somewhere, please leave a comment. I will update this post again to help others.

Happy Learning !!

Recommended Reading:

1. Equals Operator (==) vs Strict Equals Operator (===)
2. Java String hashCode() method example
3. Java String equals() method – Java compare strings
4. Always Use length() Instead of equals() to Check Empty String
5. Java Default Methods Tutorial
6. Private Methods in Interface – Java 9
7. Immutable Collections with Factory Methods in Java 9
8. Java abstract keyword – abstract classes and methods

9. Hibernate get() vs load() Methods

0. Difference between 32-bit Java vs. 64-bit Java

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

81 thoughts on “Java hashCode() and equals() Methods”

gajendra keshav bhambale

May 10, 2020 at 11:30 am

thank god someone knows how to explain thank you sir

Reply

Prashant

April 28, 2020 at 10:18 pm

The screenshot from Eclipse needs to be updated.

Reply

Lokesh Gupta

April 29, 2020 at 12:16 pm

I will update it.

Reply

Gopi

June 26, 2019 at 6:19 pm

Hi Lokesh,

I was confused when it comes to Strings, Let see the bellow code

```
package com.mytest.garbage;
public class StringTest {
  public static void main(String [] ar){
    String str = "hello";
    String str1 = new String("hello");
    if(str == str1){
      System.out.println("both are equal");
    }else{
      System.out.println("both are not  equal");
    }

    System.out.println("hashcode of str "+ str.hashCode());
    System.out.println("hashcode of str1 "+ str1.hashCode());
  }
}
```

Output:

————-

both are not equal

hashcode of str 99162322

hashcode of str1 99162322

Here hash code for both the string are same but they are not equal.Can you please help me out from this.

Thanks in advance!

Reply

Kuldeep Bisen

July 7, 2019 at 8:50 pm

As you know that two different objects can have the same hashcode. this is a case and that's the reason yo equal method here to compare the content. use == to compare primitive type, not wrapper type.

Reply

Bernard

September 22, 2019 at 2:26 pm

For String Class .equals() method and hashCode() method are already overridden. So when you compare hashcode of 2 strings with same content as in your case you will get same hashcode. But when you compare strings with == it performs reference comparisons for str and str1 in your case. Since the references are different thats why == gives false. If you compare str and str1 with .equals() you will get result as true.

Reply

sachin rokade

May 25, 2018 at 1:40 pm

why PRIME value is 31

Reply

sindhu

February 16, 2018 at 9:49 pm

Hi, your blog really helps a lot for all, I appreciate for sharing your knowledge with us.
I am having a small problem, i would be glad if you help me out.

While i am working with hashmap<String,List>, my requirement is to read data from file and set to the bean class, then i need to get the data from bean class and based on conditions i need to add to the list then add to the map. In each if- block i am adding to the map–>map.put(stringvalue,listobject), in each if-block in map entry value is getting replaced with new value and again new key value is getting returned.
like–>when i added first record in first if block
1) v->first record
when again i add to map the second record in another second if block then
2) [v—>{first record,second record},M–>{second record}]
But my requirement is like
v->{first record} M–>{second record}
and when finally i retrieve from map all the values are getting replaced with the latest values.
please suggest me with a solution.

Reply

@anonymous

March 8, 2017 at 5:11 pm

Very nice article. I understood clearly about the contract between equals and hashcode methods.

Reply

tushar

October 3, 2016 at 11:55 pm

Thanks for the elaborative article

Reply

Lakshman

July 22, 2016 at 5:28 pm

Another Important Note you can add is:

– If two objects are not equal according to

```
equals(Object)
```

then it doesn't necessarily mean that their hashcode must be different.

Reply

Raj kiran

February 1, 2016 at 4:28 am

good one.somehow i understand

Reply

Marcos

December 7, 2015 at 7:48 am

when in doubt, use constant value for hashCode

Reply

vaibhav

November 30, 2015 at 12:49 pm

Where did you find that hashCode method return memory address of an object?

Reply

Lokesh Gupta

November 30, 2015 at 2:08 pm

It's not memory address; it's integer representation of memory address.

Hint:
https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29

Please let me know if you think otherwise.

Reply

Palash Kanti Kundu

November 17, 2015 at 7:23 am

Nice Article. It helped me refreshing my knowledge on Hashcode and Equals. Please keep up the good work. i will surely check your blog time to time…

🙂

Reply

Sharda Singh

July 9, 2015 at 10:33 am

very nice and well explained. Keep posting..

Reply

Ananth

March 11, 2015 at 9:13 am

I guess it alsow returns false because both are Wrapper class.
this.getId() == e.getId()

Correct me if i am wrong!

Reply

rakesh

December 23, 2014 at 3:40 am

how two different object can have same hashcode,if they have it means they try to address two different value on same position in memory??????

Reply

Lokesh Gupta

December 23, 2014 at 7:30 am

Usually this happens when somebody override the hashCode() method.

Reply

Bhagwati Prasad

February 24, 2016 at 6:07 am

Two objects may be located in separate location on heap, but they can still be equal. But the Equals-Hashcode contract says, if two objects are equal then they must have the same hashcode.

That's why your hashcode cannot be integer representation of memory address anymore. you need to override it.

Reply

Ivan

December 4, 2014 at 11:36 am

```
@Override
public int hashCode()
{
final int PRIME = 31;
int result = 1;
result = PRIME * result + getId();
return result;
}
```

What's the purpose of result variable?

Reply

Lokesh Gupta

December 4, 2014 at 12:11 pm

I have used only “id” variable that’s why it looks like un-necessary. A complete method will look like this:

```
@Override
public int hashCode() {
  final int prime = 31;
  int result = 1;
  result = prime * result + ((department == null) ? 0 : department.hashCo
  result = prime * result + ((firstname == null) ? 0 : firstname.hashCode
  result = prime * result + ((id == null) ? 0 : id.hashCode());
  result = prime * result + ((lastName == null) ? 0 : lastName.hashCode()
  return result;
}
```

Reply

Morten

November 12, 2014 at 8:30 pm

I find it tedious to implement equals() and hashCode() by hand. I also see too many mistakes in code in this area. Better to automate this, so I have just released a free open source tool VALJOGen (valjogen.41concepts.com) which can generate your value classes with setters/getters (*), Object.hashCode, Object.equals, Object.toString, Comparable.compareTo and more from plain java interfaces. The generated output can be automatically updated when you change your interfaces – no need to maintain the generated code.

The generated output can be automatically updated when you change your interfaces so no need to maintain the generated code. Compared to alternatives it is much more customisable so no problem if you want to add your own value based custom methods, have the generated code subclass a base class with a non-default constructor, change hashCode to cache results etc. Everything can be customised.

Let me know what you think?

Reply

Anshuman Dwivedi

November 11, 2014 at 11:21 am

I've absorbed every line of article except "If you override one, then you should override the other".

Why implementations of both are required ?
What does it convey ?
What if I override only hashCode() ?

If possible then kindly explain by any scenario.

Reply

Lokesh Gupta

November 11, 2014 at 11:40 am

You must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable. – from Effective Java, by Joshua Bloch

Remember that Java puts a rule that "If two objects are equal using Object class equals method, then the hashcode method should give the same value for these two objects."

In fact, there is very are chance that you will be first modifying your hashcode() method. In most of the cases, you will need to compare two instances logically rather than default compare which compare the object references in memory. e.g. Two user objects created with new keyword will be stored in different memory locations so not equal according to default comparison using "==" operator. But for application, they actually represent same user entity and they should be treated as same instance. If one session in application says, user with id 1 is logged in, then in another part of application some code should be able to say same thing about user (1) using different instance.

To do this, you must override equals() method first of all (before even overriding hashcode()). Now, above rule says that "If two objects are equal using Object class equals method, then the hashcode method should give the same value for these two objects." If you are able to satisfy this rule then there is no need to override any more function. If rule is broken, you must override to make rule happy. You see, java has not made it mandatory. No compilation errors.

For more discussion, please read comments below. Some good questions already have been answered there.

Reply

rajesh

October 21, 2014 at 8:43 pm

I have a doubt, what happens if we override only hashcode() method.

Reply

Nagesh

October 15, 2014 at 8:05 am

“Above class prints two objects in second print statement. If both employee objects have been equal, in a Set which stores only unique objects, there must be only one instance inside HashSet, after all both objects refer to same employee. What is it we are missing??”

————————–

At this point, you could probably explain why it is printing two objects… (probably some discussion on what happens internally when you add e1 object to HasSet, e2 object to HashSet and how do you end up with 2 objects in the HashSet).

This would help add more detail about the working of the Hashing mechanism in Java.

————————–

“We are missing the second important method hashCode(). “

Reply

Lokesh Gupta

October 15, 2014 at 11:59 am

HI Nagesh, Thanks for asking a good question. I would suggest anybody who wants to know why it printed two objects : Please drill down the sourcecode of HashSet.add() method.
In short, HashSet uses a hashmap to store unique values as "keys" in map.
Now how hashmap works, is another topic you may be interested in.

Reply

Serge

September 29, 2014 at 7:18 am

Hello! Why do you use "31" number and multiplication on 1 (!) in hash code generating?
final int PRIME = 31;
int result = 1;
result = PRIME * result + getId();

If employee id is int, and hashcode is integer, you can just return and employee's id. Am I wrong?

Reply

Lokesh Gupta

September 29, 2014 at 8:00 am

A good discussion happened here: https://stackoverflow.com/questions/3613102/why-use-a-prime-number-in-hashcode

But as far as above example is concerned, you seems right. There is not much difference.

Reply

Subhankar

September 24, 2014 at 11:57 am

Excellent Explanation. Finally i understood why we override hashcode() and eqals() methods.
Thank you Sir.

Reply

swekha

September 5, 2014 at 7:19 am

in which case two objects have the same hashcode??

Reply

Lokesh Gupta

September 5, 2014 at 7:43 am

If class has overridden the hashCode() method then it is possible. e.g. Integer class. You create 10 instances of 'new Integer(10)', all will have same hashcode i.e. 10.

Reply

vinod

October 10, 2014 at 11:56 am

Then why you have written this line? "If you override one, then you should override the other."

Reply

Pallabi Kar

August 28, 2014 at 3:06 am

Very nice article.

Reply

abhi

July 26, 2014 at 7:34 pm

Nicely explained !! .. Thanks

Reply

anonymous

July 24, 2014 at 12:48 pm

this explanation is the best after a long search for a good neat example with explanation.Thankyou 🙂

Reply

Santhosh Varma

May 29, 2014 at 1:37 pm

Thanks for explaining the ORM related issue

Reply

Jin

May 29, 2014 at 10:01 am

Is it O.K to use getId() which returns an Integer and which can be null? And is there any reason that you compared the id(Integer) with a double-equal-sign?

Reply

arti

April 16, 2014 at 9:07 am

very well explained..

Reply

Hemant Sharma

April 11, 2014 at 6:05 am

Hello Sir, Tutorials are very good, only the red mark on the image is misplace, i think it should on hashCode() and equals(), but its at generate getters and setters...
You may rectify this...

Reply

Lokesh Gupta

April 11, 2014 at 8:02 am

Are you sure that you posted the comment in correct page? There is no image in this post.

Reply

Hemant Sharma

April 11, 2014 at 9:59 am

where you told about generating the hashCode and equals method using eclipse

Reply

Lokesh Gupta

April 11, 2014 at 10:23 am

Ohh.. I am sorry. I am behind some firewall which is preventing image loading from wordpress.com domain. Please ignore my comment.

Reply

HIMANSU NAYAK

April 8, 2014 at 2:58 pm

Hi Lokesh,
hashcode() of the object is only used during collections with hashing principle but if i don't have to use the object in any of the collections then is it fine to override only the equal() method.. is there any scenario where my object will behave strangely because of not overriding the hashcode().

Reply

Lokesh Gupta

April 8, 2014 at 3:01 pm

No. it will not. Everything works fine until you try to use methods where hashCode() is needed.

Reply

Sagar

March 28, 2014 at 10:36 am

A. Suppose I have a class Employee and I am trying to add the same instance of Employee, thrice to a HashSet like below:
Employee e1 = new Employee(123); //Consider 123 as EmpId.
and now:
Set sEmployee = new HashSet();
sEmployee.add(e1);
sEmployee.add(e1);
sEmployee.add(e1);

Also, I have written my equals in such that it always return true and hashcode() always returns a diff number.
1.) Will the above set contain 3 entries of the duplicate obj?
2.) What is the order in which equals() and hashcode() are called?Because ,if hashcode is diff then there is no need to even look at equals method
3.) How many times equals and hashcode will be called in above case? Please elaborate a little on this

B. How can we sort a HashMap(empid(key),empDeptName(value)) first based on empid and then based on DeptName.

Collections.sort() takes only List as parameter, so cant even send a custom comparator here.

Reply

Lokesh Gupta

March 29, 2014 at 9:53 am

Have you tried above anything yourself. I will appreciate if you try first. I am happy to discuss if something results into un-expected behavior.

Reply

Jinesh

May 6, 2014 at 11:58 pm

I believe, since you are overriding hashcode which returns different numbers for the same object it still depends on which bucket these object go to. If
1. Lets say, if hashcodes are 11, 12 and 13 but the hashset api applies another hashing function on the hashcode which determines the bucket. so if these all still fall under same bucket, then it will start checking equals which it always returns true and it will keep replacing the new object on exsiting object. The result is you will have only one bucket and that will have the last object entered. So Size = 1.
2. On the other hand, if the hashset api puts these into different buckets then you will have same objects saved thrice in different buckets and hence SIZE = 3.
3. With the same argument, SIZE can even be 2.

Guys, reply if you concur or think otherwise.

Reply

allen

March 25, 2014 at 6:02 am

thanks very much

Reply

sara

March 14, 2014 at 12:14 am

excellent!!!!!! simply superb… very clear and easy to understand..

Reply

shweta ramteke

February 28, 2014 at 5:42 am

best article after searching for so long time,i found this article the best n helpful.

Reply

Mandar

December 30, 2013 at 8:23 pm

it is awesome .. after lot of googling I find this article is the very best ..

it gives clear idea .. very good example thanks for posting this ..

Reply

sanjay

December 19, 2013 at 12:32 pm

When we are comparing string obj like String str="ABC";
str.equals("XYZ");
In this case "XYZ" is a object or not?
How it is working in default implementation of equals method.

Reply

Lokesh Gupta

December 19, 2013 at 10:06 pm

"XYZ" is also an object at lower level in JVM, So JVM interpret str.equals("XYZ"); and str.equals(new String("XYZ")); same.

Reply

Nawazish

December 18, 2013 at 11:30 pm

Hi Lokesh,

For some reason I was in a situation where I needed to customize equals() and hashCode () methods. The situation was that based on the primary key, say the employee ID of an employee, I needed to insert elements into an HashSet. However to my discomfort, I am not getting the desired behavior out of the HashSet collection. I am quoting the code below with my comments:

```
class Student
{
int id;

Student (int id)
{
this.id=id;
}

public boolean equals (Student st)
{
if (this.id==st.id)
return true;
else
return false;
}
public int hashCode ()
{
return 31*id;
}
}
```

```
class StudentSet
{
public static void main (String [] args)
{
Set set=new HashSet();
Student s1=new Student (1001);
Student s2=new Student (1001);

System.out.println ("hashCode: "+s1.hashCode());
System.out.println ("hashCode: "+s2.hashCode());

System.out.println ("is s1 Added: "+set.add (s1)); // true
System.out.println ("is s2 Added: "+set.add (s2)); // true ???

System.out.println ("equals: "+s1.equals(s2));

System.out.println ("size: "+set.size()); //size: 2 ???
}
}
```

As I re-implemented equals() for employee "id" based equality and hashCode() as well, I expected the size of the collection to be 1. However it gives "2". Could you explain the behavior please?

Reply

Lokesh Gupta

December 24, 2013 at 7:16 pm

It is because you are not overriding the equals() method correctly. Correct way is to pass Object as method argument. Best way to detect is add an @override annotation. It will tell you that method is overridden or local.

```
@Override
public boolean equals(Object st) {
if (this.id == ((Student)st).id)
return true;
else
return false;
}
```

Reply

H Singh

November 18, 2013 at 7:13 pm

Why wait(), notify() and notifyAll() are delared final in Object class ?

Reply

mani

March 4, 2014 at 8:16 am

because lock taken place on object

Reply

DH

September 27, 2013 at 4:27 am

Thanks for the post, very useful reference.

"Whenever a.equals(b), then a.hashCode() must be same as b.hashCode()" – is that "is and only if"? i.e. are there cases where hashCode() are the same but equals() should return false?

Also, I've been searching for any mentions of whether or not there's a pitfall with the following code:

```
public boolean equals( Object o )
{
// assume that hashCode() has been implemented correctly like the examples in this article
return ( o == this || ( o != null && o.getClass() == getClass() && o.hashCode() == hashCode() ) );
}
```

Basically, this enforces the contract that equals must return true if the hashCode() of two objects are the same… does it not?

Reply

DH

September 27, 2013 at 4:44 am

Nevermind, I think I answered my own question… that is, objects that are not logically equal can end up with the same hash value due to imperfect hash function – can't believe I totally forgot that 🙂

Reply

Rahul Godara

September 21, 2013 at 11:37 pm

very well explained… thank you 🙂

Reply

Prabu

September 13, 2013 at 12:02 pm

Very Useful one…thank u….

Reply

Krepil23

September 9, 2013 at 2:39 pm

Can u explain situation like master detail when id is unknown for more than one child objects (ids will be generated on persist). When initial id is 0 (if primitive) for more then one object, but I want put it in HashSet like childs, and then persist the Master with childs.

Creating hashCode from just childId is not good in that case because I'll get just one object in that childs HashSet?

For situation like this, I create hash and eqals from more than one property. Am I doing wrong?

P.S. Sry on my english :/

Reply

Lokesh Gupta

September 9, 2013 at 10:43 pm

In this specific situation, you sounds right to me.

Reply

idledevi

August 22, 2013 at 5:54 pm

Thank you for helpful information.
For whatever reason, is it ok to override hashcode() and do not override equals?
All blogs talk only about if you override equals(), then you have to override hashcode(). what about reverse?

Reply

Lokesh Gupta

August 22, 2013 at 11:55 pm

It all depends on the usecase where you will use your class.

e.g. if class instances will used as keys in HashMap then you must carefully design your class and check the behavior of both methods because both participate in put and get method in hashmap.

If you might want to implement logical comparison instead of default comparison mechanism in java, you can live happy with only overriding equals() method only.

If there is none of above scenario, then there is no need to override any of above. But if still you want to override, do at your will. It will not make much difference.

Reply

Amrit

August 21, 2013 at 10:06 pm

Thanks Lokesh for clearing out my doubts …I went through many blogs but this one was the best, giving a clear picture .

Reply

shriman

August 18, 2013 at 6:10 pm

Thanks for explaining the ORM related issue.

Reply

Tarun Snoi

August 13, 2013 at 1:56 pm

thank it clear all my doubt about hashcode() or equals () method

Reply

sujit

July 29, 2013 at 12:27 pm

Excellent !!

Reply

Jake

July 23, 2013 at 3:31 pm

I have read that the default hashcode is computed by using the class name, method names and member variables. source : hashcode internals. Is it right or not?

Reply

Lokesh Gupta

July 24, 2013 at 12:57 pm

It depends on JVM implementations of hashCode() native method.

Most implementations are like:

```
public int hashCode() {
return VMMemoryManager.getIdentityHashCode(this);
}
```

This getIdentityHashCode() method is native method. Please note that identity hashcode takes no account of the content of the object, just where it is located.

System.identityHashCode(java.lang.Object) is another method similar to VMMemoryManager.getIdentityHashCode().

I guess, I gave enough pointers to you look for more information, if still not convinced.

Reply

neeraj Patel

July 17, 2013 at 12:06 am

Very Helpful.Keep posting..

Reply

Lakhan Singh

July 15, 2013 at 11:59 am

thanks it is really very helpful …

Reply

swapna

June 5, 2013 at 2:55 am

good explanation

Reply

B

June 25, 2013 at 4:14 pm

Excellent explanation,I appreciate you very much,

Reply

Javed

February 15, 2014 at 7:41 am

i have a question that,
while calculating hashcode is it possible that we got hashcode value out of int range ?

Reply

Lokesh Gupta

February 15, 2014 at 8:07 am

No. It's not possible.

Reply

rajeshwar nagampet (@rajeshwarn433)

February 19, 2013 at 8:18 am

Well Explained... Thanks for the article...

Reply

Leave a Comment

Name *

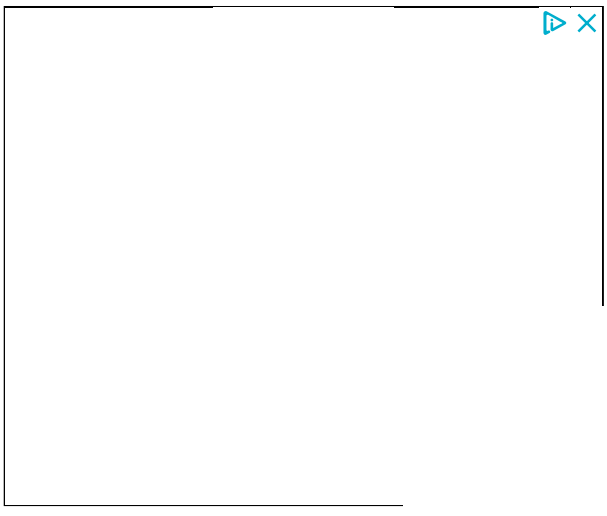
Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...



HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial