Why do I need to override the equals and hashCode methods in Java?

Recently I read through this Developer Works Document. The document is all about defining `hashCode()` and `equals()` effectively and correctly, but I am not able to figure out why we need to override these two methods.

How can I take the decision to implement these method efficiently?

java equals hashcode

edited Nov 4 '12 at 14:52
palacsint
10.8k 5 28 67

asked Feb 15 '10 at 11:17
Shashi
3,345 5 28 71

18 Answers

It is not always necessary to override hashcode and equals. But if you think you need to override one, then you need to override both of them. Let's analyze what whould happen if we override one but not the other and we attempt to use a `Map`.

Say we have a class like this and that two objects of `MyClass` are equal if their `importantField` is equal (with `hashCode` and `equals` generated by eclipse)

```
public class MyClass {

    private final String importantField;
    private final String anotherField;

    public MyClass(final String equalField, final String anotherField) {
        this.importantField = equalField;
        this.anotherField = anotherField;
    }

    public String getEqualField() {
        return importantField;
    }

    public String getAnotherField() {
        return anotherField;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                + ((importantField == null) ? 0 : importantField.hashCode());
        return result;
    }

    @Override
    public boolean equals(final Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        final MyClass other = (MyClass) obj;
        if (importantField == null) {
            if (other.importantField != null)
                return false;
        } else if (!importantField.equals(other.importantField))
            return false;
        return true;
    }
}
```

Override only `hashCode`

Imagine you have this

```
MyClass first = new MyClass("a","first");
MyClass second = new MyClass("a","second");
```

If you only override `hashCode` then when you call `myMap.put(first,someValue)` it takes first, calculates its `hashCode` and stores it in a given bucket. Then when you call `myMap.put(second,someOtherValue)` it should replace first with second as per the Map Documentation because they are equal (according to our definition).

But the problem is that equals was not redefined, so when the map hashes `second` and iterates through the bucket looking if there is an object `k` such that `second.equals(k)` is true it won't find any as `second.equals(first)` will be `false` .

Override only `equals`

If only `equals` is overriden, then when you call `myMap.put(first,someValue)` first will hash to some bucket and when you call `myMap.put(second,someOtherValue)` it will hash to some other bucket (as they have a different `hashCode`). So, although they are equal, as they don't hash to the same bucket, the map can't realize it and both of them stay in the map.

Hope it was clear

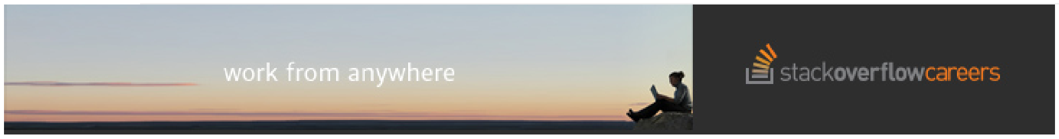
edited Aug 21 '14 at 16:27 answered Feb 15 '10 at 11:43
Lombo
2,982 1 9 23

Thanks for a nice explanation. – Code Enthusiastic Jul 25 '13 at 17:15

can you please elaborate a little more , in second case , why the second object must go in another bucket? – Hussain Akhtar Wahid 'Ghouri' May 5 '14 at 23:31

that was the best explanation of **equals and hashcode**. I would suggest it will be more appropriate if u mention in code(in comments) that how you actually want this code to work. – sagar Sep 12 '14 at 5:10

I don't like this answer because it suggests that you can't override hashCode() without overriding equals(), which is simply not true. You say your example code (the "override only hashCode" part) won't work because you *define* your two objects as equal, but - sorry - this definition is only in your head. In your first example you have two un-equal objects with the same hashCode, and that is perfectly legal. So the reason you need to override equals() is not because you have already overridden hashCode(), but because you want to move your "equals" definition from your head to the code. – user2543253 Dec 30 '14 at 15:38



> 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

By defining equals() and hashCode() consistently, you can improve the usability of your classes as keys in hash-based collections. As the API doc for hashCode explains: "This method is supported for the benefit of hashtables such as those provided by java.util.Hashtable."

The best answer to your question about how to implement these methods efficiently is suggesting you to read Chapter 3 of Effective Java.

edited Feb 15 '10 at 11:35 answered Feb 15 '10 at 11:25
JuanZe
4,843 16 43

DOWNVOTED this doens't really answer the question, it just links to a book. – bharal Dec 20 '14 at 10:32

Simply put, the equals-method in Object check for reference equality, where as two instances of your class could still be semantically equal when the properties are equal. This is for instance important when putting your objects into a container that utilizes equals and hashcode, like HashMap and Set. Let's say we have a class like:

```
public class Foo {
    String id;
    String whatevs;

    Foo(String id, String whatevs) {
        this.id = id;
        this.whatevs = whatevs;
    }
}
```

We create two instances with the same **id**:

```
Foo a = new Foo("id", "something");
Foo b = new Foo("id", "something else");
```

Without overriding equals we are getting:

- a.equals(b) is false because they are two different instances
- a.equals(a) is true since it's the same instance
- b.equals(b) is true since it's the same instance

Correct? Well maybe, if this is what you want. But let's say we want objects with the same id to be the same object, regardless if it's two different instances. We override the equals (and hashcode):

```
public class Foo {
    String id;
    String whatevs;

    Foo(String id, String whatevs) {
        this.id = id;
        this.whatevs = whatevs;
    }

    @Override
    public boolean equals(Object other) {
        if (other instanceof Foo) {
            return ((Foo)other).id.equals(this.id);
        }
    }

    @Override
    public int hashCode() {
        return this.id.hashCode();
    }
}
```

As for implementing equals and hashcode I can recommend using Guava's helper methods

edited Jul 23 '13 at 10:53
Shashi
3,345 5 28 71

answered Feb 15 '10 at 11:29
crunchdog
3,546 2 13 16

2 link unreachable – mauretto Jun 17 '13 at 14:12

Because if you do not override them you will be use the default implentation in Object.

Given that instance equality and hascode values generally require knowledge of what makes up an object they generally will need to be redefined in your class to have any tangible meaning.

answered Feb 15 '10 at 11:20
PaulJWilliams
12.1k 27 55

Joshua Bloch said it better than me in *Effective Java*, so just take a look at it !

answered Feb 15 '10 at 11:26
Valentin Rocher
7,928 21 47

I was looking into the explanation " If you only override hashCode then when you call myMap.put(first,someValue) it takes first, calculates its hashCode and stores it in a given bucket. Then when you call myMap.put(first,someOtherValue) it should replace first with second as per the Map Documentation because they are equal (according to our definition)." :

I think 2nd time when we are adding in myMap then it should be the 'second' object like myMap.put(second,someOtherValue)

answered Nov 12 '10 at 5:49
Narinder
21 1

In order to use our own class objects as keys in collections like HashMap, Hashtable etc.. , we should override both methods (hashCode() and equals()) by having an awareness on internal working of collection. Otherwise, it leads to wrong results which we are not expected.

answered Mar 11 '14 at 9:38
Prashanth
71 4

Assume you have class (A) that aggregates two other (B) (C), and you need to store instances of (A) inside hashtable. Default implementation only allows distinguishing of instances, but not by (B) and (C). So two instances of A could be equal, but default wouldn't allow you to compare them in correct way.

answered Feb 15 '10 at 11:22
Dewfy
11.8k 4 31 67

---

hashCode() :

If you only override hash-code method nothing will happen. Because it always return new hash-code for each object as an Object class.

equals() :

If you only override equal method, a.equals(b) is true it means the hash-code of a and b must be same but not happen. Because you did not override hash-code method.

Note : hash-code() method of Object class always return new hash-code for each object.

So when you need to use your object in the hashing based collection, must override both equals() and hash-code().

answered Jul 29 '13 at 11:31
Rinkal Gupta
11 1

It is useful when using **Value Objects**. The following is an excerpt from the Portland Pattern Repository:

> Examples of value objects are things like numbers, dates, monies and strings. Usually, they are small objects which are used quite widely. Their identity is based on their state rather than on their object identity. This way, you can have multiple copies of the same conceptual value object.
>
> So I can have multiple copies of an object that represents the date 16 Jan 1998. Any of

> these copies will be equal to each other. For a small object such as this, it is often easier to create new ones and move them around rather than rely on a single object to represent the date.
>
> A value object should always override .equals() in Java (or = in Smalltalk). (Remember to override .hashCode() as well.)

answered Feb 15 '10 at 11:24
Ionuț G. Stan
69.4k 12 112 149

Both the methods are defined in Object class. And both are in its simplest implementation. So when you need you want add some more implementation to these methods then you have override in your class.

For Ex: eqauls() method in object only checks its equality on the reference. So if you need compare its state as well then you can override that as it is done in String class.

answered Feb 15 '10 at 11:26
GuruKulki
8,377 16 72 135

You can find a good tutorial here regarding significance of equals and hashcode methods.
http://www.thejavageek.com/2013/06/28/significance-of-equals-and-hashcode/
http://www.thejavageek.com/2013/06/26/what-is-the-significance-of-equals-method-in-java/

answered Jul 23 '13 at 7:20
Prasad Kharkar
7,731 1 14 33

The methods equals and hashcode are defined in the object class. By default if the equals method returns true, then the system will go further and check the value of the hash code. If the hash code of the 2 objects is also same only then the objects will be considered as same. So if you override only equals method, then even though the overridden equals method indicates 2 objects to be equal , the system defined hashcode may not indicate that the 2 objects are equal. So we need to override hash code as well.

answered Jul 28 '13 at 8:06
Aarti
9 1

If the equals method returns true, there's no need to check the hashcode. If two objects have different hashcodes, however, one should be able to regard them as different without having to call equals. Further, knowledge that none of the things on a list have a particular hash code implies that none of the things on the list can match nay object with that hash code. As a simple example, if one has a list of objects whose hash codes are even numbers, and a list of objects where they are odd numbers, no object whose hash code is an even number will be in the second list. – supercat Jul 28 '13 at 19:13

If one had two objects X and Y whose "equals" methods indicated they matched, but X's hash code was an even number and Y's hash code was an odd number, a collection as described above which noted that object Y's hash code was odd and stored it on the second list would not be able to find a match for object X. It would observe that X's hash code was even, and since the second list doesn't have any objects with even-numbered hash codes, it wouldn't bother to search there for something that matches X, even though Y would match X. What you should say... – supercat Jul 28 '13 at 19:17

...would be that many collections will avoid comparing things whose hash codes would imply that they cannot be equal. Given two objects whose hash codes are unknown, it is often faster to compare them directly than compute their hash codes, so there's no guarantee that things which report unequal hash codes but return `true` for `equals` will not be regarded as matching. On the other hand, if collections happen notice that things cannot have the same hash code, they're likely not to notice that they're equal. – supercat Jul 28 '13 at 19:20

Please check the below link for proper and easy explanation: http://javapapers.com/core-java/hashcode-and-equals-methods-override/

answered Dec 4 '13 at 7:57

Abhi Sharma
11 3

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."

So, if in our class we override equals we should override hashcode method also to fallow this rule. These both methods `equals` and `hashcode` are used in `Hashtable` to store values as key-value pair.If we do override one and not the other , there is a possibility that the hashtable may not work as we want, if we use such object as key.

answered Jan 1 '14 at 15:13
Ritesh Kaushik
126 2 10

hashCode() method is used to get a unique integer for given object. This integer is used for determining the bucket location, when this object needs to be stored in some HashTable, HashMap like data structure. By default, Object's hashCode() method returns and integer representation of memory address where object is stored.

The hashCode() method of objects is used when we insert them into a HashTable, HashMap or HashSet. More about hastables on Wikipedia.org for reference.

To insert any entry in map data structure, we need both key and value. If both key and values are user define data types, the hashCode() of the key will be determine where to store the object internally. When require to lookup the object from the map also, the hash code of the key will be determine where to search for the object.

The hash code only points to a certain "area" (or list, bucket etc) internally. Since different key objects could potentially have the same hash code, the hash code itself is no guarantee that the right key is found. The HashTable then iterates this area (all keys with the same hash code) and uses the key's equals() method to find the right key. Once the right key is found, the object stored for that key is returned.

So, as we can see, a combination of the hashCode() and equals() methods are used when storing and when looking up objects in a HashTable.

NOTES:

1. Always use same attributes of an object to generate hashCode() and equals() both. As in our case, we have used employee id.
2. 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 you override one, then you should override the other.

http://parameshk.blogspot.in/2014/10/examples-of-comparable-comporator.html

answered Oct 10 '14 at 19:38
Paramesh Korrakuti
50 6

Collections such as HashMap and HashSet use the hashcode value of an object to determine how the object should be stored in the collection, and the hashcode is used again to help locate the object in the collection.

Hashing retrieval is a two-step process.

1. Find the right bucket (using hashCode())
2. Search the bucket for the right element (using equals())

here is a small example why we should overrride equals() and hashcode().Cosider a Employee class which have two fields age and name.

```
public class Employee {

    String name;
    int age;

    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == this)
            return true;
        if (!(obj instanceof Employee))
            return false;
        Employee employee = (Employee) obj;
        return employee.getAge() == this.getAge()
                && employee.getName() == this.getName();
    }

 // commented
/*  @Override
    public int hashCode() {
        int result=17;
        result=31*result+age;
        result=31*result+(name!=null ? name.hashCode():0);
        return result;
    }
*/
}
```

Now create a class, insert Employee object to a HashSet and test whether that object is present or not.

```
public class ClientTest {
    public static void main(String[] args) {
        Employee employee = new Employee("rajeev", 24);
        Employee employee1 = new Employee("rajeev", 25);
        Employee employee2 = new Employee("rajeev", 24);

        HashSet employees = new HashSet();
        employees.add(employee);
        System.out.println(employees.contains(employee2));
        System.out.println("employee.hashCode():  " + employee.hashCode()
                + "  employee2.hashCode():" + employee2.hashCode());
    }
}
you will find result like this
false
employee.hashCode():  321755204  employee2.hashCode():375890482
```

Now remove the comment from hashcode() and execute the same you will find result like this

```
true
employee.hashCode():  -938387308  employee2.hashCode():-938387308
```

Now can you see why if two objects are considered equal, their hashcodes must also be equal? Otherwise, you'd never be able to find the object since the default hashcode method in class Object virtually always comes up with a unique number for each object, even if the equals() method is overridden in such a way that two or more objects are considered equal. It doesn't matter how equal the objects are if their hashcodes don't reflect that. So one more time: If two objects are equal, their hashcodes must be equal as well.

answered Nov 27 '14 at 5:47

rajeev pani..
176 2 13

Adding to @Lombo 's answer

When will you need to override equals() ?

The default implementation of Object's equals() is

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

which means two objects will be considered equal only if they have the same memory address which will be true only if you are comparing an object with itself.

But you might want to consider two objects the same if they have the same value for one or more of their properties (Refer the example given in @Lombo 's answer).

So you will override `equals()` in these situations and you would give your own conditions for equality.

I have successfully implemented equals() and it is working great.So why are they asking to override hashCode() as well?

Well.As long as you don't use "*Hash*" *based Collections* on your user-defined class,it is fine. But some time in the future you might want to use `HashMap` or `HashSet` and if you don't `override` and "*correctly implement*" *hashCode()*, these Hash based collection won't work as intended.

Override only equals (Addition to @Lombo 's answer)

```
myMap.put(first,someValue)
myMap.contains(second); --> But it should be the same since the key are the same.But
returns false!!! How?
```

First of all,HashMap checks if the hashCode of `second` is the same as `first` . Only if the values are the same,it will proceed to check the equality in the same bucket.

But here the hashCode is different for these 2 objects (because they have different memory address-from default implementation). Hence it will not even care to check for equality.

If you have a breakpoint inside your overridden equals() method,it wouldn't step in if they have different hashCodes. `contains()` checks `hashCode()` and only if they are the same it would call your `equals()` method.

Why can't we make the HashMap check for equality in all the buckets? So there is no necessity for me to override hashCode() !!

Then you are missing the point of Hash based Collections. Consider the following :

```
Your hashCode() implementation : intObject%9.
```

The following are the keys stored in the form of buckets.

```
Bucket 1 : 1,10,19,... (in thousands)
Bucket 2 : 2,20,29...
Bucket 3 : 3,21,30,...
...
```

Say,you want to know if the map contains the key 10. Would you want to search all the buckets? or Would you want to search only one bucket?

Based on the hashCode,you would identify that if 10 is present,it must be present in Bucket 1. So only Bucket 1 will be searched !!

edited Dec 9 '14 at 6:14

answered Dec 9 '14 at 6:06
user104309
28 6