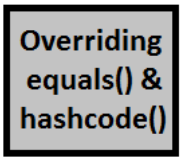
Overriding equals and hashcode method - Top 18 Interview questions and answers java for experienced and freshers



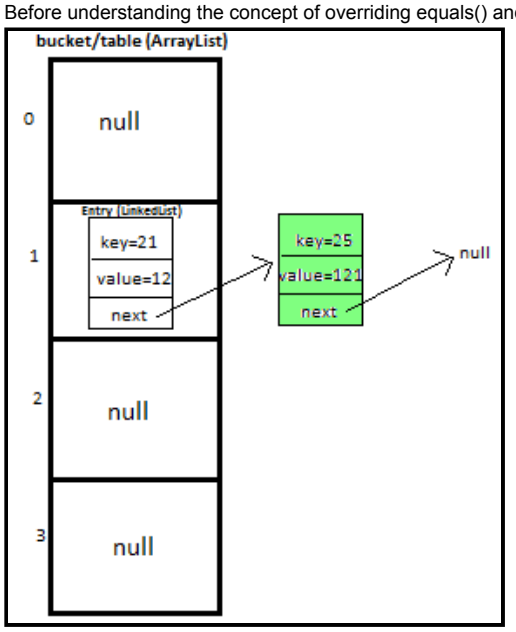
Interviewers always have great focus on checking interviewees in depth knowledge about overriding equals and hashCode() related question. So, I have tried to cover most of the possible question which could be framed related to this topic.

Question1. Why do we need to override equals and hashcode method?

Answer. Equals() and hashCode() are methods of java.lang.Object

It's important to override equals equals() and hashCode() method of class if we are want to use class as key in HashMap.
If we don't override equals() and hashCode() method we will be able to put object, but we won't be able to retrieve object.

Before understanding the concept of overriding equals() and hashCode() method, we must understand what is **bucket, Entry, and Entry.next**



Bucket is ArrayList of Entry.
Entry is LinkedList which contains information about key, value and next.
Entry.next points to next Entry in **LinkedList**.

>Why to override hashcode method?
It helps in finding bucket location, where entry(with key-value pair) will be stored .
Entry (of type **LinkedList)** is stored in **bucket (ArrayList).**

If, hashCode() method is overridden properly, we will find bucket location using hashCode() method, we will obtain **Entry** on that bucket location, then iterate over each and every E calling **Entry.next**) and check whether new and existing keys are equal or not. If keys are equal replace key-value in Entry with new one, else call Entry.next But, now the question co to check whether two keys are equal or not. So, it's time to implement equals() method.

If, hashcode method is not overridden for same key every time hashCode() method is called it might produce different hashcode, there might happen **2 cases** i.e. when **put and ge method** are called.

Case 1 : when put() method is called-
There might be possibility that same Entry (with key-value pair) will get stored at multiple locations in bucket.
Conclusion> key- value pair may get stored multiple times in HashMap.

Case 2 : when get() method is called-
As there is possibility that hashCode() method might return different hashcode & rather than searching on bucket location where Entry(with key) exists we might be searching on some other bucket location.
Conclusion> key existed in HashMap, but still we were not able to locate the bucket location in which it was stored.

>Why to override equals method?
Once we have located bucket location in which our Entry (with key-value pair) will be stored, **Equals** method helps us in finding whether **new and existing keys are equal or not**.

If we equals method is not overridden - though we will be able to find out correct bucket location if hashCode() method is overridden correctly, but still if equals method is not overri there might happen **2 cases** i.e. when **put and get method** are called.

Case 1 : when put() method is called-
we might end up storing new Entry (with new key-value pair) multiple times on same bucket location (because of absence of equals method, we don't have any way of comparing key's),

In this case, even if keys are equal, we will keep on calling Entry.next until we reach last Entry on that bucket location and ultimately we will end up storing new Entry (w
key) again in same bucket location.

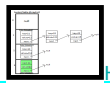
Conclusion> key- value pair stored multiple times in HashMap.

Case 2 : when get() method is called-

we won't be able to compare two keys (new key with existing **Entry**.key) and we will call **Entry**.next and again we won't be able to compare two keys and ultimately when Entr
null - we will return **false**.

Conclusion> key existed in HashMap, but still we were not able to retrieve it.

So, it's important to override equals method to check equality of two keys.

See :  HashMap Custom implementation for detailed implementation of equals and hashcode method and for understanding bucket and Entry)

See here for detailed understanding of how to put, get, remove Employee object in custom HashMap.

Question2. How do we override equals and hashcode method, write a code to use Employee as key in HashMap? (Important)

Answer. We will override equals() and hashCode() like this -

By overriding equals() and hashCode() method we could use custom object as key in HashMap.

1) Check whether obj is null or not.

```
if(obj==null) //If obj is null, return without comparing obj & Employee class.
```

2) check whether obj is instance of Employee class or not.

```
if(this.getClass()!=obj.getClass()) //identifies whether obj is instance of Employee class or not.
```

3) Then, type cast obj into employee instance.

```
Employee emp=(Employee)obj;  //type cast obj into employee instance.
```

```
@Override
public boolean equals(Object obj){

	if(obj==null)
		return false;

	if(this.getClass()!=obj.getClass())
		return false;

	Employee emp=(Employee)obj;
	return (emp.id==this.id || emp.id.equals(this.id))
			&& (emp.name==this.name || emp.name.equals(this.name));

}

@Override
public int hashCode(){
	int hash=(this.id==null ? 0: this.id.hashCode() ) +
			(this.name==null ? 0: this.name.hashCode() );
	return hash;
}
```

Let's say in an organisation there exists a employee with **id=1 and name='sam'** and **some data** is stored corresponding to him, but if modifications have to be made in data, **previ
must be overridden**.

DETAILED DESCRIPTION : Override equals() and hashCode() method.

Question3. What classes should i prefer to use a key in HashMap? (Important)

Answer. This question will check your in depth knowledge of Java's Collection Api's. we should prefer **String, Integer, Long, Double, Float, Short and any other wrapper class.**
behind using them as a key is that they override equals() and hashCode() method, we need not to write any explicit code for overriding equals() and hashCode() method.

Let's use Integer class as key in HashMap.

```
import java.util.HashMap;
import java.util.Map;

public class StringInMap {
    public static void main(String...a){

          //HashMap's key=Integer class  (Integer's api has already overridden hashCode() and
equals() method for us )
           Map<Integer, String> hm=new HashMap<Integer, String>();
           hm.put(1, "data");
           hm.put(1, "data OVERRIDDEN");

           System.out.println(hm.get(1));

    }
}
/*OUTPUT

data OVERRIDDEN

*/
```

If, we note above program, what we will see is we didn't override equals() and hashCode() method, but still we were able to store data in HashMap, override data and retrieve data usi
method.

>Let's check in **Integer's API**, how Integer class has overridden equals() and hashCode() method :

```
      public int hashCode() {
       return value;
 }

      public boolean equals(Object obj) {
       if (obj instanceof Integer) {
        return value == ((Integer)obj).intValue();
       }
       return false;
 }
```

Question4. If two objects have same hashcode, are they always equal?

Answer. **No**, It's not necessary that object's having same hashcode are always equal. Because same hashcode means object are stored on same bucket location, as key/object in
stored in Entry(Linked List), key/object's might be stored on Entry.next (i.e. on some different entry)

Question5. If two objects equals() method return true, do objects always have same hashcode?

Answer. **Yes**, two objects can return true only if they are stored on same bucket location.
First, hashCode() method must have returned same hashcode for both objects, than on that bucket location's Entry key.equals() is called, which returns true to confirm objects/keys a
So, if object's equals return true, they always have same hashcode.

Question6. Can two unequal objects have same hashcode?

Answer. **Yes**, two unequal objects can have same hashcode.

Question7. What is difference between using instanceOf operator and getClass() in equals method?

Answer. If we use **instanceOf** it will return true for comparing current class with its subclass as well,
but **getClass**() will return true only if exactly same class is compared. Comparison with any subclass will return false.

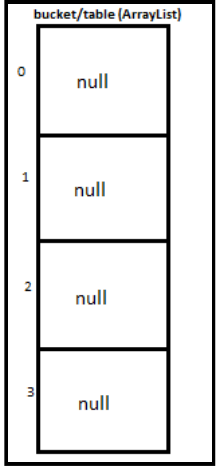
Question8. How we can implement HashMap for using Integer as key? (Important)

Answer. **Here** Interviewer tends to check your knowledge of overriding equals and hashCode method, and also how Api's use Integer internally as key in HashMap.

Let's understand in detail how Integer is used as key in Hashmap.

Initially, we have bucket of **capacity=4.** (all indexes of bucket i.e. 0,1,2,3 are pointing to null).

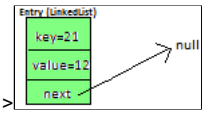
Note: Initial capacity provided by Java Api is different.



Let's put first key-value pair in HashMap-

Key=21, value=12

newEntry Object will be formed like this >

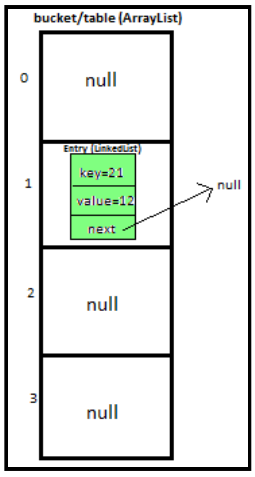


We will calculate hash by using our **hash(K key)** method **-** in this case it returns **key/capacity= 21%4= 1**.

So, **1** will be the **index of bucket** on which **newEntry object** will be stored.

We will go to **1st** index as it is pointing to null we will **put our newEntry object there**.

At completion of this step, our HashMap will look like this-



DETAILED DESCRIPTION : How Integer is used as key in Hashmap

Question9. Implement custom HashMap to put, get, remove Employee object

Answer. See HashMap Custom implementation - put, get, remove Employee object

dri bolgEdit

Interviewers sometimes tend to confuse interviewees by framing different question by **overriding either of equals() or hashCode()** metho
Interviewers aims to check **in depth knowledge** of **interviewees** that how many **buckets** are formed, what is **sizeOf HashMap** after each expe
and **get()** method returns object or not **(Important)**.

Question10. How many **buckets** will be there and what will be **size of HashMap**?

```
package p1;

import java.util.HashMap;
```

```
class Employee {

    private String name;

    public Employee(String name) { // constructor
           this.name = name;
    }


    //no hashCode() method

    //no equals() method

    @Override
    public String toString() {
           return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program1 {
    public static void main(String...a){

           HashMap<Employee, String> hm=new HashMap<Employee, String>();
           hm.put(new Employee("a"), "emp1");
           hm.put(new Employee("b"), "emp2");
           hm.put(new Employee("a"), "emp1 OVERRIDDEN");

           System.out.println("HashMap's data> "+hm);
           System.out.println("HashMap's size> "+hm.size());
           System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
/*OUTPUT

HashMap's data> {Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=a] =emp1, Employee[ name=b] =emp2}
HashMap's size> 3
null

*/
```

Buckets= As hashCode() method is not there, hashcode generated for 3 objects will be different and we will end up using **3** buckets.

Size= As equals() method is not their, size will be **3**.

get()=we won't be able to get object.

Question11. How many buckets will be there and what will be size of HashMap?

```
package p2;

import java.util.HashMap;

class Employee {

    private String name;

    public Employee(String name) { // constructor
           this.name = name;
    }

    @Override
    public int hashCode(){
           return (this.name==null ? 0: this.name.hashCode() );
    }

    @Override
    public boolean equals(Object obj){
           Employee emp=(Employee)obj;
           return (emp.name==this.name || emp.name.equals(this.name));
    }

    @Override
    public String toString() {
           return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program2 {
```

```
    public static void main(String...a){

            HashMap<Employee, String> hm=new HashMap<Employee, String>();
            hm.put(new Employee("a"), "emp1");
            hm.put(new Employee("b"), "emp2");
            hm.put(new Employee("a"), "emp1 OVERRIDDEN");

            System.out.println("HashMap's data> "+hm);
            System.out.println("HashMap's size> "+hm.size());
            System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
/*OUTPUT

HashMap's data> {Employee[ name=b] =emp2, Employee[ name=a] =emp1 OVERRIDDEN}
HashMap's size> 2
emp1 OVERRIDDEN

*/
```

Buckets= As hashCode() method is overridden perfectly, **2** bucket locations will be used.

Size= As equals() method is their, size will be **2**,
value corresponding to `Employee with id=1 and name='sam'` was **`employee1 data___`**
& was overridden by value **`employee1 data OVERRIDDEN_`**

get()=we will be able to get object.

Question12. How many buckets will be there and what will be size of HashMap?

```
package p3;

import java.util.HashMap;

class Employee {

    private String name;

    public Employee(String name) { // constructor
            this.name = name;
    }

    @Override
    public int hashCode(){
            return 1;
    }

    @Override
    public boolean equals(Object obj){
            return true;
    }

    @Override
    public String toString() {
            return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program3 {
    public static void main(String...a){

            HashMap<Employee, String> hm=new HashMap<Employee, String>();
            hm.put(new Employee("a"), "emp1");
            hm.put(new Employee("b"), "emp2");
            hm.put(new Employee("a"), "emp1 OVERRIDDEN");

            System.out.println("HashMap's data> "+hm);
            System.out.println("HashMap's size> "+hm.size());
            System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
/*OUTPUT

HashMap's data> {Employee[ name=a] =emp1 OVERRIDDEN}
HashMap's size> 1
emp1 OVERRIDDEN
```

```
*/
```

Buckets= As hashCode() method returns 1, only **1** bucket location will be used.

Size= As equals() method always returns true, size will be **1**, all three employees will be stored on same bucket location in one Entry (new Entry will keep on overriding previous Entry always get last stored key-value pair only.

get()=we will be able to get object.

Question13. How many buckets will be there and what will be size of HashMap?

```
package p4;

import java.util.HashMap;

class Employee {

    private String name;

    public Employee(String name) { // constructor
           this.name = name;
    }

    @Override
    public int hashCode(){
           return 1;
    }

    //no equals() method

    @Override
    public String toString() {
           return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program4 {
    public static void main(String...a){

           HashMap<Employee, String> hm=new HashMap<Employee, String>();
           hm.put(new Employee("a"), "emp1");
           hm.put(new Employee("b"), "emp2");
           hm.put(new Employee("a"), "emp1 OVERRIDDEN");

           System.out.println("HashMap's data> "+hm);
           System.out.println("HashMap's size> "+hm.size());
           System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
 /*OUTPUT

HashMap's data> {Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=b] =emp2, Employee[ name=a] =emp1}
HashMap's size> 3
null

*/
```

Buckets= As hashCode() method returns 1, only **1** bucket location will be used.

Size= As equals() method doesn't exist, size will be **3**, all three employees will be stored on same bucket location but in different Entry.

get()=we won't be able to get object.

Question14. How many buckets will be there and what will be size of HashMap?

```
package p5;

import java.util.HashMap;

class Employee {

    private String name;

    public Employee(String name) { // constructor
           this.name = name;
    }
```

```
    //no hashCode() method

    @Override
    public boolean equals(Object obj){
           return true;
    }

    @Override
    public String toString() {
           return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program5 {
    public static void main(String...a){

           HashMap<Employee, String> hm=new HashMap<Employee, String>();
           hm.put(new Employee("a"), "emp1");
           hm.put(new Employee("b"), "emp2");
           hm.put(new Employee("a"), "emp1 OVERRIDDEN");

           System.out.println("HashMap's data> "+hm);
           System.out.println("HashMap's size> "+hm.size());
           System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
/*OUTPUT

HashMap's data> {Employee[ name=b] =emp2, Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=a] =emp1}
HashMap's size> 3
null

*/
```

Buckets= As hashCode() method is not there, hashcode generated for 3 objects will be different and we will end up using **3** buckets.

Size= Though equals() method is their(but because of hashCode() method's absence) **which always returns true**, we won't be able to locate correct bucket location for calling equals method, so, size will be **3**.

get()=we won't be able to get object.

Question15. How many buckets will be there and what will be size of HashMap?

```
package p6;

import java.util.HashMap;

class Employee {

    private String name;

    public Employee(String name) { // constructor
           this.name = name;
    }

    @Override
    public int hashCode(){
           return (this.name==null ? 0: this.name.hashCode() );
    }

    //no equals() method


    @Override
    public String toString() {
           return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program6 {
    public static void main(String...a){

           HashMap<Employee, String> hm=new HashMap<Employee, String>();
           hm.put(new Employee("a"), "emp1");
           hm.put(new Employee("b"), "emp2");
           hm.put(new Employee("a"), "emp1 OVERRIDDEN");
```

```
            System.out.println("HashMap's data> "+hm);
            System.out.println("HashMap's size> "+hm.size());
            System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
/*OUTPUT

HashMap's data> {Employee[ name=b] =emp2, Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=a] =emp1}
HashMap's size> 3
null

*/
```

Buckets= As hashCode() method is overridden perfectly, **2** bucket locations will be used.

Size= As equals() method is not their, size will be **3**,

get()=we won't be able to get object.

Question16. How many buckets will be there and what will be size of HashMap?

```
package p7;

import java.util.HashMap;

class Employee {

    private String name;

    public Employee(String name) { // constructor
            this.name = name;
    }

    //no hashCode() method

    @Override
    public boolean equals(Object obj){
            Employee emp=(Employee)obj;
            return (emp.name==this.name || emp.name.equals(this.name));
    }

    @Override
    public String toString() {
            return "Employee[ name=" + name + "] ";
    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class Program7 {
    public static void main(String...a){

            HashMap<Employee, String> hm=new HashMap<Employee, String>();
            hm.put(new Employee("a"), "emp1");
            hm.put(new Employee("b"), "emp2");
            hm.put(new Employee("a"), "emp1 OVERRIDDEN");

            System.out.println("HashMap's data> "+hm);
            System.out.println("HashMap's size> "+hm.size());
            System.out.println(hm.get(new Employee("a")));

    }
}
```

Answer.

```
/*OUTPUT

HashMap's data> {Employee[ name=a] =emp1, Employee[ name=a] =emp1 OVERRIDDEN, Employee[ name=b] =emp2}
HashMap's size> 3
null

*/
```

Buckets= As hashCode() method is not there, hashcode generated for 3 objects will be different and we will end up using **3** buckets.

Size= Though equals() method is their(but because of hashCode() method's absence), we won't be able to locate correct bucket location for calling equals() method, so, size will be **3**

get()=we won't be able to get object.