**HashCode and Equals method in Java object**

Introduction:

Every Class extends Object class implicitly so that they provide basic features which according to Java recommendation every class should have. Such as clone(), equals(), hashCode(), toString(), etc.

\*\*java.lang.Object\*\* is a base class of every class in java. This class also provides lots of methods in it with its custom implementation and two of them are \*\*\***equals()**\*\*\* and \*\*\***hashCode()**\*\*\* and both of the methods play very important role in big and small application both.

Let’s have a look how we provide our custom implementation for both.

\*\*\***equals() :**\*\*\* if we want to compare two primitive values we use ‘= =’ operators but what if we want to compare two object will it work for them also ??

of course not we can not compare two objects using ‘= =’ operator because ‘= =’ always compare the location where they are stored ie. the reference of both of the objects.

In below images we can see object s1 and s2 are shared at 10001 and 10002 which reference or the stored location respectively of both of the objects.

Lets have a

``` java

class Student{

int name;

int age;

String sex;

getters..

Setters…

}

```

s1

s2

10001

10002

Fig:1.2

s1==s2 means 10001==10002

s1.name.equals(s1.name)

Fig:1.3

Student s1=new Student();

s1.name=”sample name”;

s1.age=21;

s1.sex=”male”;

Student s2=new Student();

s2.name=”sample name”;

s2.age=21;

s2.sex=”male”;

Fig:1.1

**Fig 1: equals and ==**

![im1 equals and ==]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im1\_eagerload.png)

In fig:1 we can see that we have two object s1 and s2 which are stored in memory which can be seen in fig:2.

In fig:3 we can clearly see that if we check equality of object s1 and s2 using ‘= =’ it will check if the reference id of both of the object are same or not.’= =’ is used when we check equality of primitives but in case of objects we use \*\*\*equals()\*\*\* method.

Below is an example through which we can understand how to give our own implementation for equals method to compare two objects.

``` java

@Override

public boolean equals(Object obj){

if(obj==null) return false;

if(!(obj instanceof Student)) return false;

if(obj == this) return true;

return this.getId() == ((Student) obj).getId();

}

```

Above code snippet confirms that the object coming for the comparison are equal if their id’s are same.

As soon as will deep dive in to the explanation things would become more clear.

So stay tuned …

\*\*\***hashCode() :**\*\*\* Java provides a number of data structures to store our data in collection. For example, several Map interface implementations are available to store our data in key and value pair which internally use HashTable.

hashMap(s1,”student 1”);

Key Value

Get hashcode from key

Evaluate array index from hashcode

Eg: array index came is 4

0 1 2 3 4 5 6 7 8 9 10 ….

`

**Fig 2 : hashmap to array**

![im2 hashmao to array]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im2\_hashmaptoarray.png)

When using hashtable to store our data the internal mechanism of java uses hashcode which is calculated on the basis of your implementation of hashCode() method in your code.

Simply put hashCode() and return the integer value based on your algorithm written in your hashCode() method.

Objects that are equal must return same hashcode but \*\*\***Its not required for different objects to return different hashcode**\*\*\*.

\*\*General contract for hashcode() method\*\*

* Whenever it is invoked on the same object more than once during an execution of a Java application, hashCode() must consistently return the same value, provided no information used in equals comparisons on the object is modified. This value needs not remain consistent from one execution of an application to another execution of the same application
* If two objects are equal according to the equals(Object) method, then calling the hashCode() method on each of the two objects must produce the same value
* It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, developers should be aware that producing distinct integer results for unequal objects improves the performance of hash tables

Hashcode plays very important role when it comes to Map Collection. In map collection when we save our data using \*\*\***key\*\*\*** and **\*\*\*value\*\*\***

Map<Student,String> studentMap=new HashMap<Student,String>();

Student s=new Student();

s.name=”Piyush”;

s.age=25;

s.sex=”male”;

studentMap.put(s,”student 1”);

Name: Piyush

Age:25

Sex:Male

Student 1

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

**Fig 3 : Kay and Value**

![im3 key and value]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im3\_keyandvalue.png)

and internally what happens is using \*\*\***key**\*\*\* a method hashCode() generates hash code which is nothing but a integer value of the location/index of an array.

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

**Fig 4 : hashcode method**

![im4 hashcode method]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im4\_hashcode.png)

After getting that integer value of the location of an array will store the pointer of one linked list.

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

returning 6

6

1001

Array index

1001

Name: Piyush

Age:25

Sex:Male

Student 1

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

**Fig 5 : moving to array and linked list**

![im5 moving to array and linkedlist]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im5\_movingtoaarayandll.png)

we will be keep saving our map objects likewise

Name: Sameer

Age:26

Sex:Male

Student 2

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

Array index

returning 7

7

1002

1001

1002

1001

**Fig 6 : saving more elements in map**

![im6 saving more elements in map]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im6\_savingmoreele.png)

**###What is hash collision**

All way all good but what happens is when hash collision occurs. when we get repeated hash code. Say we have two different objects but the hashcode for both of the objects are same. Here hash collision comes in to the picture. So what happens when two object gets same hashcode they get same array index and both of the objects get saved into the linkedlist whose pointer is saved at that array index.

Name: Sameer

Age:26

Sex:Male

Student 2

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Name: Samar

Age:46

Sex:Male

Student 3

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Hash collision

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

Array index

returning 7

7

1002

6

1003

1001

1002

1001

**Fig 7 : hash collision**

![im7 hash collision]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im7\_hashcollision.png)

Since we already have one object whose value is “Student 2” and again we get one more entry of object whose value is “Student 3”. We can see this in above diagram.

“Student 2” and “Student 3” both are returning same hash value and this is called hash collision.

So internally what happens is our values get stored in linked list where each node contains \*\*\***value**\*\*\*

**### Getting values from hashmap**

While getting values from hashmap we primarily perform two operation.

**\*\*\*Step1.\*\*\*** Using \*\*hashCode()\*\* we get hashcode from the key and go to the respective location of array to get the corresponding value from linkedlist on that array index.

**\*\*\*Step2.\*\*\*** After reaching to the linkedlist using \*\*equals()\*\* we compare/check equality for each element available on that list with current our key whose corresponding value we need to fetch.

Name: Sameer

Age:26

Sex:Male

Student 2

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Name: Samar

Age:46

Sex:Male

Student 3

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

Array index

7

1002

6

1003

1001

1002

1001

Step 2

Step 1

**Fig 8 : fetching elements from map**

![im8 fetching elements from map]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im8\_fetchingelement.png)

Public statuc void main(String a[]){

Map<Student,String> m=new HashMap();

Student s1=new Student();

s1.name=”samar”;

s1.age=46;

s1.sex=”Male”;

Student s2=new Student();

s2.name=”sameer”;

s2.age=26;

s2.sex=”Male”;

m.put(s1,”student 1”);

m.put(s2,”student 2”);

Student s=new Student();

s.name=”samar”;

s.age=46;

s.sex=”Male”;

m.get(s);

}

**Fig 9 : map sample program**

![im9 map sample program]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im9\_mapsampleprog.png)

**Step1 (getting hashcode):**

Here in above diagram we can see that we have one psvm method and we are creating three Student object and s1,s2 and s respectively. Till s1 and s2 we have already discussed in above section but what happen at s. how do we get element from key s.

We will follow step1 first and calculate hashcode for the key. Our key is s which nothing but one object of Student. The hashCode method of class Student will be called and hashCode would be generated.

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

**Fig 10 : hashcode implemantation**

![im10 hashcode implementation]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im10\_hashcodeimpl.png)

Here hashCode() would return 31\*46 = 1426 and we are taking last digit of incoming number ie 6 so 6 is the array index where from we will get the address of one linked list and then we will move to step2 for further operation.

**Step2 (checking eqaulity):**

After reaching to linkedlist now compiler will check equality of our key with each element present in linkedlist and the element with which our key would match will contain our value.

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

public boolean equals(Object o){

Student s=(Student)o;

If(this.name.equals(s.name))

return true;

else

return false;

}

**Fig 11 : equals implementation**

![im11 equals implementation]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im11\_equalsimpl.png)

Name: Sameer

Age:26

Sex:Male

Student 2

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Name: Samar

Age:46

Sex:Male

Student 3

Age:25

Sex:Male

**Value**

Age:25

Sex:Male

**Key**

StudentMap

Student Class

..

..

..

public int hashCode(){

return 31\*this.age;

}

Array index

7

1002

6

1003

1001

1002

1001

Step 2

Step 1

**Fig 12 : getting element from map**

![im12 getting element from map]( https://github.com/PiyushMittl/java/blob/master/java\_equalsANDhashcode\_im12\_getelementmap.png)

**\*\*\*Check 1.\*\*\***  At node 1002 it will check if object at node 1002 is equals to the key we have ie

this.name ie “samar” equals to “Sameer” - - > fasle

``` java

public boolean equals(Object o){

Student s=(Student)o; 9719777430- molchand

If(this.name.equals(s.name))

return true;

else

return false;

}```

\*\*\***Check 2.**\*\*\* Move to next node at node 1003 it will check if object at node 1003 is equals to the key we have ie

this.name ie “samar” equals to “sameer” - - > true

the value “student 3” at node “1003” will be returned.