**Why to override equals() and hashCode()?**

**How i can implement both equals() and hashCode() for Set ?**

As mentioned before, if two objects are equal as defined by Equals(), they must have the same hashCode(), but that's not necessarily true the other way around. If your objects don't follow this contract, you'll get undesired behavior when storing into hash related data structures like HashMap.   
  
Say you have a 'student' class with a string name and int age. We think that two 'student' s are equal if they have the same instance variable values like name and age, therefore we override Equals. If we neglect to override hashCode() too and rely on Object()'s implementation, we will get multiple entries in our hashMap for two "equal" students(see Object() javadoc for this). If you follow the contract by overriding hashCode(), the "equal" students will hash to the same value and therefore will not appear as separate entries in your map.

------------------------------------------------------------------------------------------------------------------------------------

Some really good answers here: know how/why hashing is used in the JVM (i.e., data structures), Collections and default/non-default Object handling, not always required to override, prevent bugs, etc. A simpler, practical example may help. Changing from a standard object, say String or Number, in a Collection to your own non-standard Object in the same or similar Collection.

In the first case, you need not implement hashcode (equals) as it's already implemented correctly for you.

In the second case, you generally need to override equals and hashcode, accommodating the JVM rules as Juned explains. A common pattern I often see begins by putting non-standard Objects into a Map with a standard Object key, like String. Typical. Sometime later, a requirement changes and you change the key to a non-standard object, but overlook the need to override equals/hashcode to meet your needs. This is an easy problem to create in practice. A modern IDE plugin could warn you when overrides are not found when non-standard Objects are used in Collections. Has anyone seen something like this in Intellij or Eclipse?

If you are implementing HashSet to store unique object then you need to implement equals() and hashcode() method.

if two objects are equal according to the equals() method, they must have the same hashCode() value (although the reverse is not generally true).

Two scenarios

Case 1) : If you don't implement equals() and hashcode() method :

When you are adding objects to HashSet , HashSet checks for uniqueness using

equals() and hashcode() method the class ( ex. Emp class). If there is no equals() and hashcode() method the Emp class then HashSet checks default Object classes

equals() and hashcode() method.

In the Object class , equals method is

public boolean equals(Object obj) {

return (this == obj);

}

Under this default implementation, two references are equal only if they refer to the exact same object. Similarly, the default implementation of hashCode() provided by Object is derived by mapping the memory address of the object to an integer value.

This will fail to check if two Emp object with same employee name .

For Example :

Emp emp1 = new Emp();

emp1.setName("sat");

Emp emp2 = new Emp();

emp2.setName("sat");

Both the objects are same but based on the default above method both objects are dirrefent because references and hashcode are different .

So in the HashSet you can find the duplicate emp objects.

To overcome the issue equals() and hashcode() method need to override.

Case 2) : If you override equals() and hashcode() method

Example : implement equals and hashcode

public class Emp {

private long empId;

String name = "";

public boolean equals(Object o) {

if (o == this)

return true;

if (!(o instanceof Emp))

return false;

Emp emp = (Emp)o;

return emp. empId == empId &&

emp. name == name;

}

public int hashCode(){

int result = 10;

result = result \* new Integer(String.valueOf(empId)).intValue();

return result;

}

}

In the equals() , it check for name is same or not. This way you can find out the objects are equals or not.

In the hashCode() also it return some unique value for each object. In this way if two Emp object has same empId then it will say both are same object.

Now HashSet store only unique objects.

If you do

Emp emp1 = new Emp();

emp1.setName("sat");

Emp emp2 = new Emp();

emp2.setName("sat");

if(emp1.equals(emp2)){

System.out.println("equal");

}

This will print : equal