Equals() method and "==" operator difference:

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

"==" operator can compare both primitive type and object. it ll compare based on memory reference of two objects.

equals() method is there in java.lang.Object class. it ll compare only objects.it ll compare objects based on the business logic we written for equals method. Default implemetation of equals() method in Object class is similar to "==" equality operator.

Need of overriding equlas and hashcode method:

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

equals and hashcode method are mainly used in hash based collections for eg: hashmap and hashtable.

the general contract of hashcode method:

1.Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer,

2.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 integer result.

3.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.

we are using hashcode method to find a bucket(array location) in array for storing object in collections. as per the contract, two different object can have the same hashcode and same object should have the same hashcode. in that scenario we will use equals mathod to store the object. if equals method return true the old obejct will be replaced by the new one. otherwise linked list datastructure will come into the picture. new object will be added at the end of linked list structure.

what if we doesnot override the equals and hashcide method:

JVM assigns unique hashcode value to each object when they are created in memory and if developers don’t override the hashcode method then there is no way the two object returns same hashcode value.

Duplicate objects are added in Hashmap as a key (Because we have not overided the hashcode and equals method)

We are not able to get back object from map (Because hashcode is not implemented)

why do we need to override hashcode method if we override equals method:

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

public class CustomerID {

private long crmID;

private int nameSpace;

public CustomerID(long crmID, int nameSpace) {

super();

this.crmID = crmID;

this.nameSpace = nameSpace;

}

public boolean equals(Object obj) {

return this.crmID == ((CustomerID) obj).crmID &&

this.nameSpace == ((CustomerID) obj).nameSpace;

}

public static void main(String[] args) {

Map m = new HashMap();

m.put(new CustomerID(2345891234L,0),"Jeff Smith");

System.out.println(m.get(new CustomerID(2345891234L,0)));

}

}

Compile and run the above code, the output result is

null

What is wrong? The two instances of CustomerID are logically equal according to the class's equals method. Because the hashCode() method is not overridden, these two instances' identities are not in common to the default hashCode implementation. Therefore, the Object.hashCode returns two seemingly random numbers instead of two equal numbers. Such behavior violates "Equal objects must have equal hash codes" rule defined in the hashCode contract.

Autoboxing and unboxing:

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

it was introduced in java 1.5 to automatically convert primitive type to object or wrapper class.

when we assign a primitive value to a non-primitive type or when we pass a prmitive value to the merhod which has the non primitive type arguments:

before java 1.5 we need to convert them to object and then oly we can do the assignment or method invacation.

after java 1.5, java compiler will do the conversion automatically.

(1) Compiler uses valueOf() method to convert primitive to Object and uses intValue(), doubleValue() etc to get primitive value from Object.

2) During autoboxing boolean is converted to Boolean, byte to Byte, char converted to Character, float changes to Float, int goes to Integer, long goes to Long and short converts to Short, while in unboxing opposite happens like Float to float.