[**Difference between equals method and "==" operator in Java - Interview Question**](https://javarevisited.blogspot.com/2012/12/difference-between-equals-method-and-equality-operator-java.html)

Both equals() and "==" operator in Java is used to compare objects to check equality but the main differences between equals method and  the == operator are:

1. Equals() is a method and == is an operator. Since [Java doesn’t support operator overloading](http://javarevisited.blogspot.sg/2011/08/why-java-does-not-support-operator.html), == behaves identical for every object but equals() is method, which can be overridden in Java and logic to compare objects can be changed **based upon business rules**.
2. Another notable difference between == and equals() is that == is used to compare both primitive and objects while equals() is only used for objects comparison.
3. Another difference between them is that, If both "==" and equals() is used to compare objects than == returns true only if both references points to same object while equals() can return true or false based on its [overridden implementation](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). One of the popular cases is [comparing two String in Java](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html) in which case == and equals() method return different results.

**What is "==" equality operator in Java**

"==" or equality operator in Java is a binary operator used to compare primitives and objects. In terms of comparing primitives like boolean, int, float "==" works fine but when it comes to comparing objects it creates confusion with equals() method in Java. "==" compare two objects based on memory reference. so "==" operator will return true only if two object references it is comparing represent or refer to exactly the same object otherwise "==" will return false.



{After the introduction of [Autoboxing and unboxing in Java 5](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html), using == to compare wrapper objects even become trickier because sometimes they can return an unexpected result. See my post [what is the problem with == operator in autoboxing world post-Java 5](http://javarevisited.blogspot.sg/2010/10/what-is-problem-while-using-in.html) for more details.}

**What is equals() method in Java**

equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic. You can [**override equals() method**](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) in your class defining a condition on which two objects of your class will be considered equal. equals() has contracted with hashcode() method in Java and whenever you override equals method you also need to [**override hashcode()** in Java](http://javarevisited.blogspot.sg/2011/10/override-hashcode-in-java-example.html). Default implementation of equals() provided in Object class is similar to "=="  operator and returns true if you are comparing two references to the same object.

It’s one of the Java best practice to override equals in Java to define equality based on business requirement. It’s also worth noting that [**equals() should be consistent with compareTo() in Java**](http://javarevisited.blogspot.sg/2011/11/how-to-override-compareto-method-in.html), So that when you store objects in [TreeMap](http://javarevisited.blogspot.sg/2011/12/treemap-java-tutorial-example-program.html) or [TreeSet](http://javarevisited.blogspot.sg/2012/11/difference-between-treeset-hashset-vs-linkedhashset-java.html) Collection, which uses compareTo() for checking equality, behavior remains consistent.

**Comparing String with == and equals()**

Since java.lang.String class override equals() method, its implementation is that it returns true if two String object contains same content; but == will only return true if two references are pointing to the same object.

**Comparing two objects with "==" and equals.**

Another scenario which creates confusion between == and equals method is when you compare two Objects. When you compare two references pointing to an object of type Object you should see the **same result from both == operator** and equals method because *default implementation of equals* method just compare memory address of two objects and return true if two reference variable are pointing towards an exactly same object. Here is example of == vs equals() method for comparing two objects:

**Example**

**package** coredemo;

**class** Test {

**int** a;

**boolean** b;

**public** Test(**int** a, **boolean** b) {

**super**();

**this**.a = a;

**this**.b = b;

}

}

**public** **class** CoreDemo {

**public** **static** **void** main(String[] args) {

Test t1, t2;

t1 = **new** Test(23, **true**);

t2 = **new** Test(23, **true**);

System.***out***.println(t1 == t2);

System.***out***.println(t1.equals(t2));

String s1 = "abc";

String s2 = **new** String("abc");

System.***out***.println("Checking with String");

System.***out***.println(s1 == s2);

System.***out***.println(s1.equals(s2));

}

}

**Output:**

false

false

Checking with String

false

true

Because user defined Test class doesn’t have any equals() method defined it will go to up its hierarchy to its first super class which has defined equals(). In this case the only parent class of this Test is Object class and so its equals() is used which compares the reference and not the values of the objects so this will give false with == as well as with equals().

Because String class has equals() internally implemented to match the content of the String objects and not the reference, == will give false and equals() will give true. Note here that the Strings s1 and s2 are created differently, one with ‘new’ and one without. This ensures that both objects point to different String objects i.e. there are actually two objects created. If we used ‘= abc’ in both then only one String object in String pool would be created (only if it did not already exist in the String pool, if it existed then both of the references s1, s2 would simply point to that ‘abc’) and both s1 and s2 would refer to same object.

**Summary**

1) use == to compare primitive e.g. boolean, int, char etc., while use equals() to compare objects in Java.

2) == return true if two reference are of same object. Result of equals() method depends on overridden implementation.

3) For comparing String use equals() instead of == equality operator.