[***10 points about instanceof operator in Java with example***](http://javarevisited.blogspot.com/2015/12/10-points-about-instanceof-operator-in-java-example.html)

*The****instanceof****operator in Java is used to check if an object belongs to a particular type or not at runtime. It's also a built-in keyword in Java programming language and mostly used to*[*avoid ClassCastException in Java*](http://javarevisited.blogspot.com/2012/12/how-to-solve-javalangclasscastexception-java.html)*. It is used as safety-check before casting any object into a certain type. This operator has a form of object instanceof Type and returns****true****if the object satisfies IS-A relationship with the Type i.e. object is an instance of class Type or object is the instance of a class which extends Type or object is an instance of a class which implements interface Type. Once an object passed the instanceof check, it's safe to*[*type-cast*](http://javarevisited.blogspot.com/2012/12/what-is-type-casting-in-java-class-interface-example.html)*into that type, without worrying of****java.lang.ClassCastException****. By the way instanceof operator is also a rather lesser used operator in Java, similar to*[*transient*](http://java67.blogspot.com/2012/08/what-is-transient-variable-in-java.html)*and*[*volatile*](http://java67.blogspot.com/2012/08/what-is-volatile-variable-in-java-when.html)*, and many programmers don't even know basics of this operator. If you are one of those then you must read this article and bookmark it for future reference.*

***10 things about instanceof operator in Java***

*Good knowledge of instanceof operator is essential for Java developers, but It's, even more, important for those JEE developers, who are working in Hibernate, as instanceof has a big role to play in*[*overriding equals() and hashCode()*](http://javarevisited.blogspot.com/2015/01/why-override-equals-hashcode-or-tostring-java.html)*of Hibernate persistent class. Since Hibernate proxies everything, it's not possible to use the****getClass()****method for type checking, you need to rely on instanceof operator for that. You can also expect few questions from instanceof operator in*[*OCPJP exam*](http://javarevisited.blogspot.com/2015/11/5-free-ocajp8-ocpjp8-mock-exams-and-practice-questions.html)*or any*[*core Java interview*](http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html)*.  
  
1) instanceof operator will return false if the object is null. For example in the following code, control will not go inside if() block if object's value is null.*

*if(object instanceof Order){*

*Order ord = (Order) object;*

*}*

*You can use this property of****instanceof****operator while*[*overriding equals() method in Java*](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html)*.  
  
  
2) The****instanceof****operator is primarily used to check whether an object is of a certain type or not. In fact, the instanceof operator provides a way to perform runtime type identification in Java, which is unlike C++, not supported by Java. Alternatively, you can also use the getClass() method from java.lang.Class for the same purpose. See the article*[*2 ways of runtime type identification in Java*](http://javarevisited.blogspot.com/2012/09/how-to-determine-type-of-object-runtime-identification.html)*for more details.  
  
  
3) One of the most prominent use of****instanceof****operator is to implement****equals()****method for Hibernate persistent class. Since Hibernate uses a proxy object, it's not safe to do a type check in equals() method using getClass() method, as it will never succeed. Other ways to doing type check e.g. Hibernate.getClass() method will make your entity class dependent on third party library and if you want to share your persistent class than you also need to include Hibernate, which is not good at all. That's the reason, I prefer instanceof operator for doing type checking in the*[*equals() method of Entity class*](http://javarevisited.blogspot.com/2011/02/how-to-write-equals-method-in-java.html)*, as shown below:*

*@Override*

*public boolean equals(Object object) {*

*// instanceof also takes care of null check*

*if (!(object instanceof Customer)) {*

*return false;*

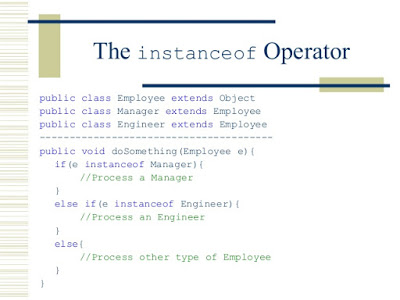
*}*

*Customer other = (Customer) object;*

*// rest of code omitted for brevity*

*}*

*Since instanceof operator in Java returns****true****for instance of a subclass, this code will work fine with Hibernate proxy classes, which inherit from the persistent class. By the way, instanceof check also breaks*[*equals() method's symmetrical contract*](http://javarevisited.blogspot.com/2013/08/10-equals-and-hashcode-interview.html)*, according to that if****a.equals(b)****is****true****than****b.equals(a)****should also be****true****, but with the instanceof operator in use, if a represent instance of parent class, and b being instance of child class, a.equals(b) will return true but b.equals(a) will return false, thus breaking equals() contract of symmetry. Though it's not a real issue, until you are doing such comparison e.g. in the*[*Hibernate persistent class*](http://javarevisited.blogspot.com/2013/05/10-hibernate-interview-questions-answers-java-j2ee-senior.html)*.*

*[](http://2.bp.blogspot.com/-mm9P0FbP8Zg/VnllkemJKXI/AAAAAAAAEZo/u7eBDgy-HOs/s1600/instanceof+operator+in+Java+with+example.jpg)*

***Java instanceof Example***

*Here is a sample Java Program to demonstrate the capability of instanceof operator in Java. First we have created a couple of classes and interfaces to define type hierarchy. We have a class called ObjectOriented and an interface JVMLanguage. Then we have another class called Java which extends ObjectOriented and implements JVMLanguageand another class called CPlusPlus which only extends the ObjectOriented class. Now with this setup, we will test the****instanceof****operator to determine whether an object is an instance of any of these classes or not.*

*class ObjectOriented {*

*}*

*class Java extends ObjectOriented implements JVMLanguage {*

*}*

*class CPlusPlus extends ObjectOriented {*

*}*

*interface JVMLanguage {*

*}*

*import java.io.File;*

*import java.io.IOException;*

*import java.util.Scanner;*

*/\*\**

*\**

*\* @author Javin Paul*

*\**

*\*/*

*public class InstanceOfDemo {*

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

*ObjectOriented oops = new ObjectOriented();*

*Java java = new Java();*

*CPlusPlus cpp = new CPlusPlus();*

*// true - because oops is direct instance of ObjectOriented class*

*System.out.println("oops instanceof ObjectOriented : "*

*+ (oops instanceof ObjectOriented));*

*// true - because java is instance of Java class,*

*which is sub-class or ObjectOriented*

*System.out.println("java instanceof ObjectOriented : "*

*+ (java instanceof ObjectOriented));*

*// true - because java is object of Java class,*

*which implements JVMLanguage interface*

*System.out.println("java instanceof JVMLanguage : "*

*+ (java instanceof JVMLanguage));*

*// true - because cpp is object of CPlusPlus,*

*which extends ObjectOriented*

*System.out.println("cpp instanceof ObjectOriented : "*

*+ (cpp instanceof ObjectOriented));*

*// true - because cpp is object of type CPlusPlus*

*System.out.println("cpp instanceof CPlusPlus: "*

*+ (cpp instanceof CPlusPlus));*

*// false - because oops is instance of ObjectOriented,*

*neither extends Java class, nor implement Java interface*

*System.out.println("oops instanceof Java : "*

*+ (oops instanceof Java));*

*//Java fromOops = (Java) oops; // compile ok*

*//- classCastException at runtime*

*// false - because ObjectOriented class does not implement*

*JVMLanguage interface*

*System.out.println("oops instanceof JVMLanguage : "*

*+ (oops instanceof JVMLanguage));*

*//JVMLanguage jvm = (JVMLanguage) oops; // compile fine*

*// - java.lang.ClassCastException at runtime*

*// false - because CPlusPlus class does not implement*

*JVMLanguage interface*

*System.out.println("cpp instanceof JVMLanguage : "*

*+ (cpp instanceof JVMLanguage));*

*// instanceof operator returns false if compared to null*

*java = null;*

*System.out.println("(java =null) instanceof ObjectOriented : "*

*+ (java instanceof ObjectOriented));*

*System.out.println("(java =null) instanceof Java : "*

*+ (java instanceof Java));*

*}*

*}*

*Output:*

*oops instanceof ObjectOriented : true*

*java instanceof ObjectOriented : true*

*java instanceof JVMLanguage : true*

*cpp instanceof ObjectOriented : true*

*cpp instanceof CPlusPlus: true*

*oops instanceof Java : false*

*oops instanceof JVMLanguage : false*

*cpp instanceof JVMLanguage : false*

*(java =null) instanceof ObjectOriented : false*

*(java =null) instanceof Java : false*

*You can also run this program by uncommenting certain code which fails instanceof check and subsequent cast will throw java.lang.ClassCastException. This program is self-explanatory, but you should pay attention when instanceof return false, you can clearly see that it returns true only when we have IS-A relationship satisfied between A and B in check of A instanceof B. If A is null or there is no IS-A relationship, it will return false and any attempt to cast B into A will throw ClassCastException.  
  
That’s all about****what is the instanceof operator in Java****and****how to use instanceof operator****. In defensive programming, instanceof check is must before casting any unknown object. By default don't use the****instanceof****operator for type checking in equals(), as it breaks the symmetry contract, but remember to use it while overriding equals() for Hibernate persistent class.*

*Read more:*[*http://javarevisited.blogspot.com/2015/12/10-points-about-instanceof-operator-in-java-example.html#ixzz3xKIozeZn*](http://javarevisited.blogspot.com/2015/12/10-points-about-instanceof-operator-in-java-example.html#ixzz3xKIozeZn)