**instanceof operator**

1. [The instanceof operator](http://www.javatpoint.com/downcasting-with-instanceof-operator#instanceof)
2. [Example of instanceof operator](http://www.javatpoint.com/downcasting-with-instanceof-operator#instanceofex)
3. [Applying the instanceof operator with a variable the have null value](http://www.javatpoint.com/downcasting-with-instanceof-operator#instanceofnull)
4. [Downcasting with instanceof operator](http://www.javatpoint.com/downcasting-with-instanceof-operator#instanceofdowncasting)
5. [Downcasting without instanceof operator](http://www.javatpoint.com/downcasting-with-instanceof-operator#instanceofdowncastingwithout)

The **instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof operator is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that have null value, it returns false.

**Simple example of instanceof operator**

Let's see the simple example of instance operator where it tests the current class.

1. class Simple{
2. public static void main(String args[]){
3. Simple s=new Simple();
4. System.out.println(**s instanceof Simple**);//true
5. }
6. }

Output:true

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

**Another example of instanceof operator**

1. class Animal{}
2. class Dog extends Animal{//Dog inherits Animal
3.
4. public static void main(String args[]){
5. Dog d=new Dog();
6. System.out.println(d instanceof Animal);//true
7. }
8. }

Output:true

**instanceof operator with a variable that have null value**

If we apply instanceof operator with a variable that have null value, it returns false. Let's see the example given below where we apply instanceof operator with the variable that have null value.

1. class Dog{
2. public static void main(String args[]){
3. Dog d=null;
4. System.out.println(d instanceof Dog);//false
5. }
6. }

Output:false

**Downcasting with instanceof operator**

When Subclass type refers to the object of Parent class, it is known as downcasting. If we perform it directly, compiler gives Compilation error. If you perform it by typecasting, ClassCastException is thrown at runtime. But if we use instanceof operator, downcasting is possible.

1. Dog d=new Animal();//Compilation error

If we perform downcasting by typecasting, ClassCastException is thrown at runtime.

1. Dog d=(Dog)new Animal();
2. //Compiles successfully but ClassCastException is thrown at runtime

**Possibility of downcasting with instanceof operator**

Let's see the example, where downcasting is possible by instanceof operator.

1. class Animal { }
2.
3. class Dog extends Animal {
4. static void method(Animal a) {
5. if(a instanceof Dog){
6. Dog d=(Dog)a;//downcasting
7. System.out.println("ok downcasting performed");
8. }
9. }
10.
11. public static void main (String [] args) {
12. Animal a=new Dog();
13. Dog.method(a);
14. }
15.
16. }

Output:ok downcasting performed

**Downcasting without the use of instanceof operator**

Downcasting can also be performed without the use of instanceof operator as displayed in the following example:

1. class Animal { }
2. class Dog extends Animal {
3. static void method(Animal a) {
4. Dog d=(Dog)a;//downcasting
5. System.out.println("ok downcasting performed");
6. }
7. public static void main (String [] args) {
8. Animal a=new Dog();
9. Dog.method(a);
10. }
11. }

Output:ok downcasting performed

Let's take closer look at this, actual object that is referred by a, is an object of Dog class. So if we downcast it, it is fine. But what will happen if we write:

1. Animal a=new Animal();
2. Dog.method(a);
3. //Now ClassCastException but not in case of instanceof operator

**Understanding Real use of instanceof operator**

Let's see the real use of instanceof keyword by the example given below.

1. interface Printable{}
2. class A implements Printable{
3. public void a(){System.out.println("a method");}
4. }
5. class B implements Printable{
6. public void b(){System.out.println("b method");}
7. }
8.
9. class Call{
10. void invoke(Printable p){//upcasting
11. if(p instanceof A){
12. A a=(A)p;//Downcasting
13. a.a();
14. }
15. if(p instanceof B){
16. B b=(B)p;//Downcasting
17. b.b();
18. }
19.
20. }
21. }//end of Call class
22.
23. class Test{
24. public static void main(String args[]){
25. Printable p=new B();
26. Call c=new Call();
27. c.invoke(p);
28. }
29. }

Output: b method