Java instanceof

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

The instanceof in java 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 has null value, it returns false.

Simple example of java instanceof

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

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(s instanceof Simple1);//true
 }
}
```

Test it Now

```
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 java instanceof operator

```
class Animal{}
class Dog1 extends Animal{//Dog inherits Animal

 public static void main(String args[]){
 Dog1 d=new Dog1();
 System.out.println(d instanceof Animal);//true
 }
}
```

Test it Now

```
Output:true
```

instanceof in java 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.

```
class Dog2{
 public static void main(String args[]){
  Dog2 d=null;
```

```
  System.out.println(d instanceof Dog2);//false
 }
}
```

Test it Now

```
Output:false
```

Downcasting with java 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.

```
Dog d=new Animal();//Compilation error
```

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

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

Possibility of downcasting with instanceof

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

```
class Animal { }


class Dog3 extends Animal {
  static void method(Animal a) {
    if(a instanceof Dog3){
      Dog3 d=(Dog3)a;//downcasting
      System.out.println("ok downcasting performed");
    }
  }
```

```
  public static void main (String [] args) {
    Animal a=new Dog3();
    Dog3.method(a);
  }

 }
```

Test it Now

```
Output:ok downcasting performed
```

Downcasting without the use of java instanceof

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

```
class Animal { }
class Dog4 extends Animal {
  static void method(Animal a) {
       Dog4 d=(Dog4)a;//downcasting
       System.out.println("ok downcasting performed");
  }
   public static void main (String [] args) {
    Animal a=new Dog4();
    Dog4.method(a);
  }
}
```

Test it Now

```
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:

```
Animal a=new Animal();
Dog.method(a);
```

```
//Now ClassCastException but not in case of instanceof operator
```

Understanding Real use of instanceof in java

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

```
interface Printable{}
class A implements Printable{
public void a(){System.out.println("a method");}
}
class B implements Printable{
public void b(){System.out.println("b method");}
}

class Call{
void invoke(Printable p){//upcasting
if(p instanceof A){
A a=(A)p;//Downcasting
a.a();
}
if(p instanceof B){
B b=(B)p;//Downcasting
b.b();
}

}
}//end of Call class

class Test4{
public static void main(String args[]){
Printable p=new B();
Call c=new Call();
c.invoke(p);
}
```

```
}
```

Test it Now

```
Output: b method
```

<<prev next>>

Share this page

Latest 4 Tutorials