Java instanceof

The **java instanceof operator** is used to test whether the object is an instance of the specified type

**1. class**

**2. subclass**

**3. interface**

**A parent object is not an instance of Child.**

**A child object is an instance of parent.**

**A parent reference referring to a Child is an instance of Child.**

**1. Class**

**class** Simple1{

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

 Simple1 s=**new** Simple1();

 System.out.println(s **instanceof** Simple1);//true

 }

}

2. Subclass

**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

 }

}

**//for null**

Dog2 d=**null**;

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

4. **A parent object is not an instance of Child**

Parent pobj = new Parent();

         if (pobj instanceof Child) //return false

**5. A parent reference referring to a Child is an instance of Child**

Parent cobj = new Child();

if (cobj instanceof Child) //true

## **Downcasting**

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

1. **class** Animal { }
3. **class** Dog3 **extends** Animal {
4. **static** **void** method(Animal a) {
5. **if**(a **instanceof** Dog3){
6. Dog3 d=(Dog3)a;//downcasting
7. System.out.println("ok downcasting performed");
8. }
9. }
11. **public** **static** **void** main (String [] args) {
12. Animal a=**new** Dog3();
13. Dog3.method(a);
14. }
16. }