Java

Qué es y cómo utilizar instanceof en Java

By Equipo Geek - NTT Data on 26-Abril, 2021 - 10:32

LINKEDIN | EMAIL | TWITTER | FACEBOOK

Estoy seguro que en algún momento hemos necesitado saber si un objeto es de un determinado tipo o no. Cuando trabajamos con objetos es importante poder controlar esto para evitar "sorpresitas desagradables" durante la ejecución de nuestro código.

Existe una palabra clave especial en Java que se encarga justamente de esto. Hablamos del operador `instanceof`, por supuesto pero antes de que dés saltos de alegría "¡Iuuiii parsa!" y no te lancen todavía. En este artículo vamos a hablar de sus pros pero también de sus contras. O dudas de esto modo, por qué instanceof debería ser tu último recurso.

Sobre el papel usar este operador es tan útil como tentador. Su sintaxis sería algo como el siguiente ejemplo:

```
obj instanceof Object
```

Al lado izquierdo tenemos la **instancia** y el lado derecho el **nombre de la clase** y como resultado el **operador** `instanceof` devuelve un resultado booleano. Con este resultado (true o false, verdadero o falso) podemos determinar si el objeto al que apunta una referencia dada es una instancia de una clase o interfaz concretas. Pero para entenderlo mejor veamos algunos ejemplos:

Ejemplos de uso de instanceof en Java

Uso de instanceof con clases

```
public class Main {
    public static void main (String[] args) {
        Perro perro = new Perro();
        System.out.print(perro instanceof Animal);   // true
    }
}

class Animal {}
class Perro extends Animal {}
class Robot {}
```

Supersencillo. El objeto al que apuntra la variable perro es claramente una instancia de la clase Perro, que a su vez hereda características de la clase Animal. Por ello, el resultado es Verdadero (true).

Ahora vayamos con el siguiente ejemplo:

```
class Animal {}
class Fish extends Animal {
  void swim[]
    System.out.println("Swim");
  }
}
class Bird extends Animal {
  void fly[]{
    System.out.println("Fly");
  }
}
class Kangaroo extends Animal {
  void jump[]
    System.out.println("Jump");
  }
}

public final class RuEnunOfInstanceOf {
  public static void main(String[] args){
    makeOfMove(new Fish());
    makeOfMove(new Bird());
    makeOfMove(new Kangaroo());
  }

  public static void makeOfMove(Animal animal){
    if (animal instanceof Fish){
      Fish fish = (Fish)animal;
      fish.swim();
    }
    else if (animal instanceof Bird){
      Bird bird = (Bird)animal;
      bird.fly();
    }
    else if (animal instanceof Kangaroo){
      Kangaroo kangaroo = (Kangaroo)animal;
      kangaroo.jump();
    }
  }
}
```

Si te han dado cuenta, entre-es un ejemplo de-un **mal uso de** `instanceof`. Me explico. Tenemos que tener en cuenta que estamos haciendo la implementación de este-mal-uso porque queremos declarar diferentes tipos con sus comportamientos correspondientes. El mal-uso que aquí-los objetos se tratan como si fuerran del **tipo de superclase** y encima no es una abstracción ¿Ni porque no contiene suficiente información para determinar qué se desea hacer con el objeto y eso puede ser un problema porque `instanceof` puede devolver valor verdadero aún cuando no disponemos de una referencia real dado que lo correcto y esperable es que sea al objeto el que apunta la referencia y no de la referencia en sí.

Esto se debe a que la referencia en sí-si-es-nada, es como un hilo que sirve de guía para llegar al objeto. La referencia en sí-no es instancia de nada. Esto te tiene que quedar muy claro, porque es posible que, de otro modo, pienses que `instanceof` devuelve siempre el valor true si se utiliza del modo antes mostrado-si en el ejemplo y estarás-mos cayendo en una trampa. Si todavía tienes dudas volvamos al primer ejemplo pero cambiándolo un poco:

```
public class Main {
    public static void main (String[] args) {
        Perro perro = new Perro();
        perro = null;
        System.out.print(perro instanceof Animal);   // false
    }
}

class Animal {}
class Perro extends Animal {}
class Robot {}
```

Quizá así lo podemos ver más claro. Aquí la referencia perro no apuntra a ningún objeto por lo que tenemos un valor null como salida, y nulo-es instancia de-nada (false).

Esto-nos deja clara la evidencia de que el resultado-no siempre nos ayudará a mantener nuestro código lo más claro y limpio posible. Sintonces ¿qué alternativas tenemos a `instanceof`. Bueno, si no queremos liliar nuestro código de-finiter quizá la mejor aproximación sea que, si queremos que el comportamiento específico dependa del tipo de objeto, entonces ese comportamiento debe encapsularse en el objeto mismo.

Uso de instanceof con interfaces

Normalmente si cometemos algún error el compilador suele devolver un mensaje de error que nos advierte del problema, pero cuando usamos el de `instanceof` con **interfaces instanceof** siempre compila. Veamos un ejemplo:

```
interface Vehicle { }
class Car { }
class Ferrari extends Car implements Vehicle {}
```

Si verificamos la palabra clave `instanceof` con los objetos de clase anteriores, veremos los siguientes resultados:

```
Ferrari Ferrari = new Ferrari();

Ferrari instanceof Vehicle // true - Ferrari implementa Vehicle
Ferrari instanceof Car     // true - Ferrari extiende de Car
Ferrari instanceof Ferrari // true - Ferrari es Ferrari
Ferrari instanceof Object  // true - El objeto es el tipo principal de todos los objetos
```

Pero si revertimos-o algo, veremos resulta-dos diferentes:

```
Car car = new Car();
car instanceof Ferrari // false - El coche es un supertipo de Ferrari, no al revés
```

instanceof vs isInstance()

Se suele recomendar utilizar el método `isInstance()` en lugar de `instanceof` para comprobar la clase del objeto. Ambos son muy parecidos, ambos devuelven un valor booleano, pero la gran diferencia es que el método de clase `isInstance()` funcionará tal-y como esperamos en cualquier escenario, algo que ya queda claro que con instanceof no-es siempre-fiable.

Veamos un ejemplo

```
public class Test
{
        public static void main(String[] args)
        {
                Integer i = new Integer(5);

                // prints true as i is instance of class
                // Integer
                System.out.println(i instanceof Integer);
        }
}

Output = True
```

Ahora, si queremos verificar la clase del objeto en tiempo de ejecución, debemos-o usar el método `isInstance()`:

```
public class Test
{
        // This method tells us whether the object is an
        // instance of class whose name is passed as a
        // string
        public static boolean Fun(Object obj, String s)
                                        throws ClassNotFoundException
        {
                return Class.forName(s).isInstance(obj);
        }

        // Driver code that calls Fun()
        public static void main(String[] args)
                                        throws ClassNotFoundException
        {
                Integer i = new Integer(5);

                // print true as i is instance of class
                // Integer
                boolean b = Fun(i, "java.lang.Integer");

                // print false as i is not instance of class
                // String
                boolean b1 = Fun(i, "java.lang.String");

                /* print true as i is also instance of class
                Number as Integer class extends Number
                class */
                boolean b2 = Fun(i, "java.lang.Number");

                System.out.println(b);
                System.out.println(b1);
                System.out.println(b2);
        }
}

Output:
true
false
true
```

NOTA: El operador `instanceof` arroja un **error de tiempo de compilación** ("incompatible conditional operand types") si verificamos el objeto con otras clases que no son-o instanciables:

```
public class Test
{
        public static void main(String[] args)
        {
                Integer i = new Integer(5);

                // Below line causes compile time error--
                // Incompatible conditional operand types
                // Integer and String
                System.out.println(i instanceof String);
        }
}

Output :

15: error: incompatible types: Integer cannot be converted to String
        System.out.println(i instanceof String);
```

Conclusión

Y llegamos al final de nuestro artículo de hoy. Pero antes hagamos un resumen de lo aprendido. El operador instanceof puede parecer una buena idea sobretodo cuando comenzamos-a-programar, pero como hemos-visto debe ser-a-usado con cautela y debería ser nuestro último recurso. Su excesivo uso puede indicar una-de-los siguientes problemas:

- Puedes estar usando-u-una-mala abstracción. Se excede-a más-de-lo que debería.
- Si se utiliza instanceof con una interfaz, siempre compila.
- Si se utiliza instanceof con una clase, sólo devuelve false-er dos casos. En el resto de casos, o devuelve true, o no compila.
- Puede-que acabes por intentar unificar cosas que pertenecen a diferentes ámbitos-en donde no existe un punto-de abstracción útil-entre las entidades-cuo-las que opera.

En lugar de `instanceof` lo recomendable es utilizar el **método de clase Java** `isInstance()`, que al igual que `instanceof` devolverá un valor booleano con la diferencia de que su resultado será mucho más fiable.

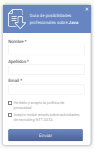
Java | Frontend

Liferay, un "Todo en uno" de No Code
por Juan Manuel Gos - 15 Mar 2022

Web scraping con javascript y nodejs
por Equipo Geek - 5 - 27 Mayo 2021

Machine Learning con Java. Fundamentos básicos
por Equipo Geek - 9 - 28 Abril 2021

Leer más



This website uses cookies

We use cookies to personalise content and ads, to provide social media features and to analyse our traffic. We also share information about your use of our site with our social media, advertising and analytics partners who may combine it with other information that you've provided to them or that they've collected from your use of their services.

Allow all | Allow selection | Deny

Necessary | Preferences | Statistics | Marketing | Show details

Powered by Cookiebot by Usercentrics