equals y hashCode en Java

Joselu Cantero

Introducción

En la clase java.lang.Object (y por lo tanto, o mejor dicho, y por herencia, en todas las demás clases) tenemos dos métodos que suelen ser dos grandes olvidados:

public boolean equals(Object obj)

public int hashCode()

Estos métodos son especialmente importantes si vamos a guardar nuestros objetos en cualquier tipo de colección: listas, mapas (aquí es especialmente importante el método hashCode), … y más aun si los objetos que vamos a guardar en la colección son serializables.

Para que nos hagamos una idea de la importancia que tienen, muchos puristas consideran que una clase no está correctamente implementada si no tiene sobreescritos estos métodos.

Comparar dos objetos con ==

La comparación mediante == entre objetos sirve para comparar si ambos objetos apuntan a la misma referencia. No sirve para comparar su contenido:

Creamos dos objetos aparentemente idénticos:

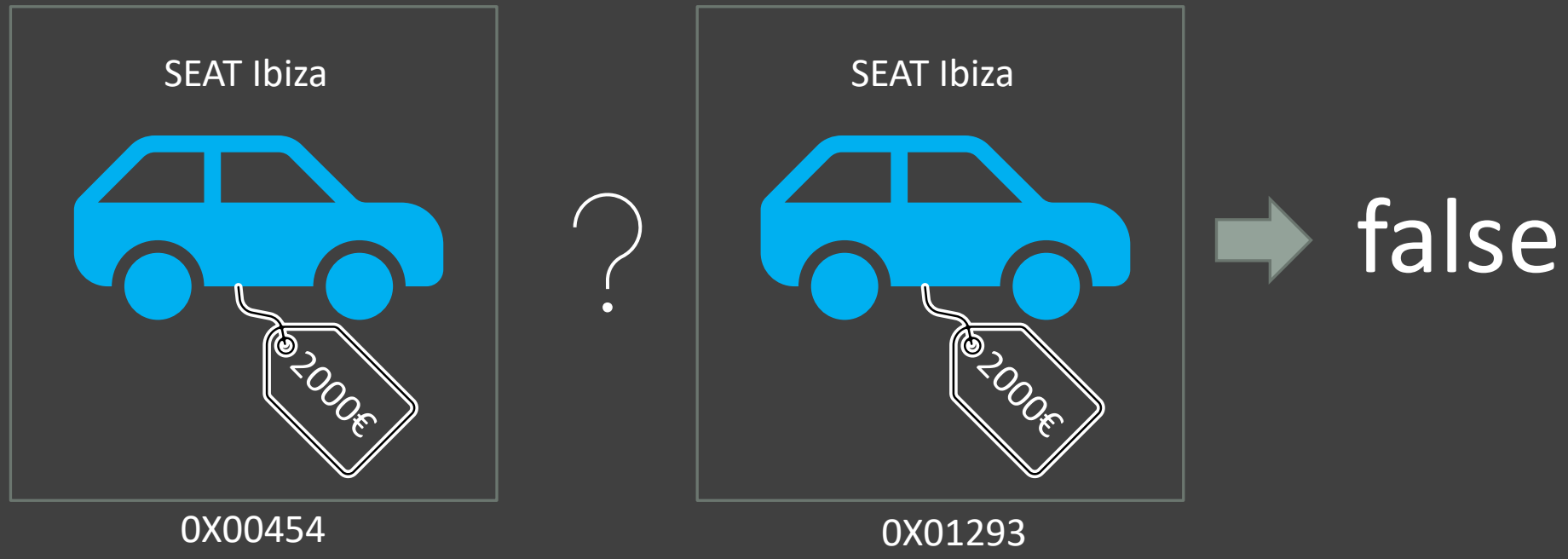
```
Automovil miAuto1 = new Automovil ("SEAT", "Ibiza", "Azul", 2000.0);

Automovil miAuto2 = new Automovil ("SEAT", "Ibiza", "Azul", 2000.0);
```

¿Es miAuto1 igual a miAuto2?

```
boolean resultado = miAuto1 == miAuto2; // ➔ false
```

Comparar dos objetos con ==



Los objetos son aparentemente iguales, pero con el operador == lo que comparamos es la referencia guardada en las variables miAuto1 (0X00454) y miAuto2 (0X01293) que son evidentemente diferentes.

Comparar dos objetos con equals

La comparación mediante equals entre objetos, por defecto, tiene un resultado… idéntico!:

¿Es miAuto1 igual a miAuto2?

```
boolean resultado = miAuto1.equals( miAuto2); // ➔ false
```

Si comprobamos la implementación del método equals en la clase Object entenderemos el porqué…

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

Sobreescribiendo equals

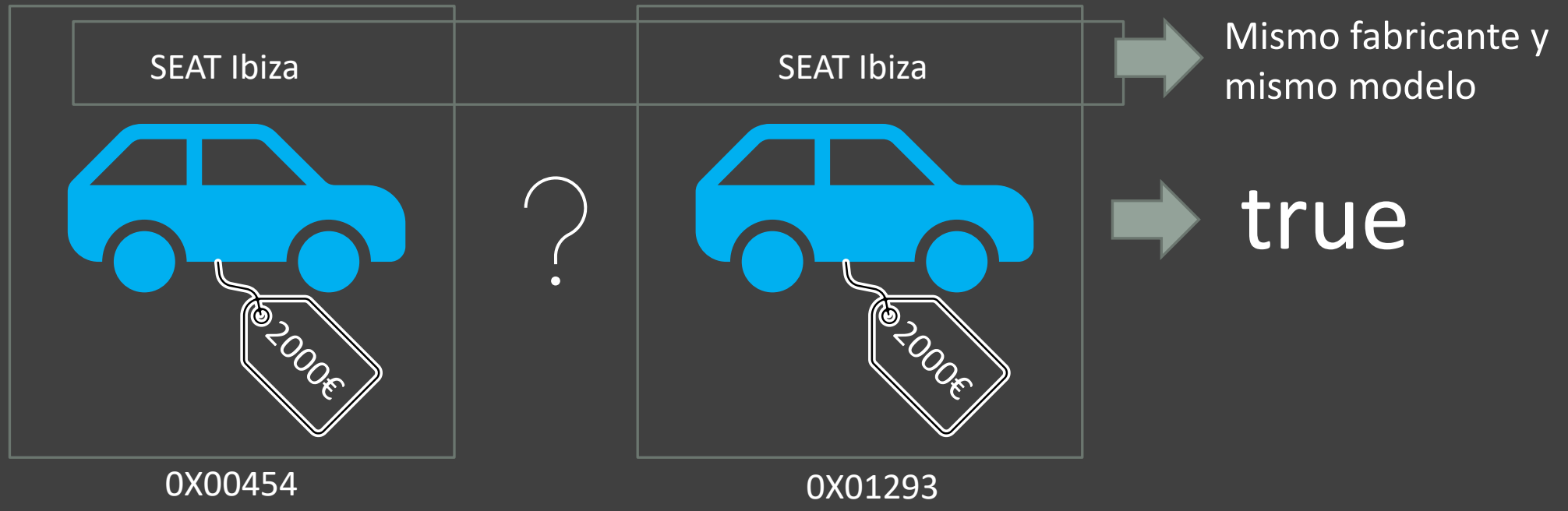
Hemos comprobado que el método equals, por defecto, utiliza ==. Pero podemos sobreescribir el método para, por ejemplo, considerar iguales dos coches del mismo fabricante y del mismo modelo Es decir crear unas condiciones de igualdad a la carta:

```
@Override
   public boolean equals(Object obj) {
        if (this == obj) return true; //Lo primero es comprobar si obj "soy yo"
        if (obj == null) return false; //Comprobamos que obj tenga referencia

      //Como recibimos un tipo genérico Object nos asegurarnos de que obj es de tipo Automovil
      if (!(obj instanceof Automovil)){ return false; }
      //En caso positivo hacemos un cast...
      Automovil autoC = (Automovil)obj;

       return this.fabricante != null && this.modelo != null && this.fabricante.equals(autoC.getFabricante()) &&
       this.modelo.equals(autoC.getModelo()); //Utilizamos el método equals de los String...
   }
```

Sobreescribiendo equals



Ahora, después de haber elegido el criterio de igualdad y haberlo implementado en el método equals (sobreescrito) ambos coches son iguales.

El método hashCode

Este método sirve para comparar objetos dentro de una estructura de tipo Hash (HashMap, HashSet, etc.). Su reescritura no es indispensable, pero sí recomendable. Una vez que sobreescribamos el método equals tenemos la obligación de sobreescribir el método hashCode.

El método hashCode() devuelve un dato de tipo int que identifica al objeto. Dicho entero debería estar en función de valores (a la carta) que determinen cuando un objeto es igual o distinto de otro, en nuestro ejemplo instancias de la clase Automóvil.

```
@Override
public int hashCode() {
        return Objects.hash(this.fabricante , this.modelo );
}
```

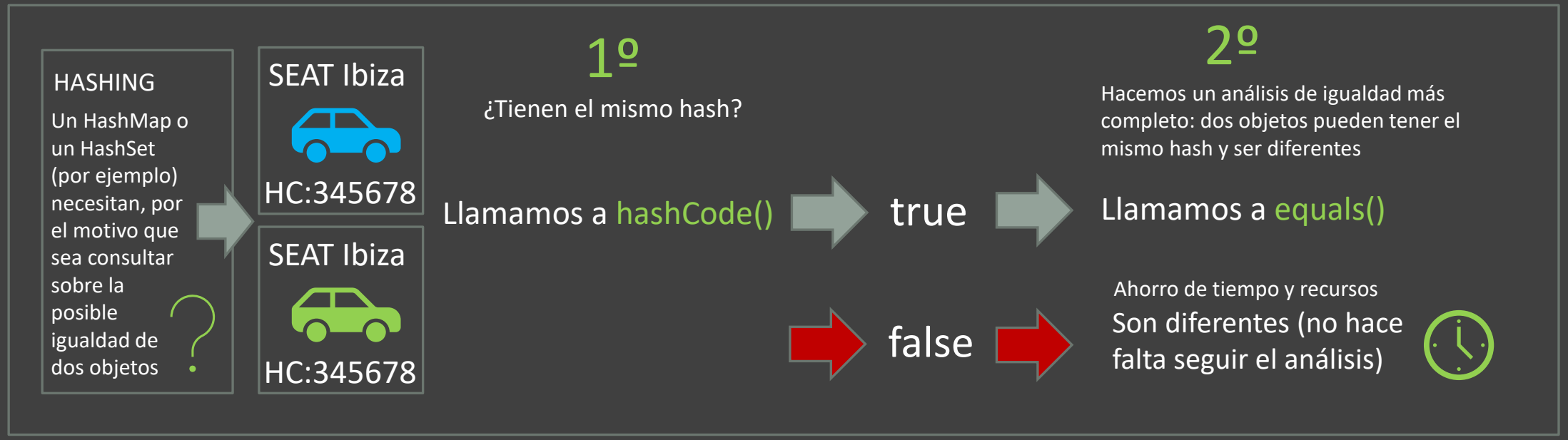
Podemos comprobar cómo dos instancias de automóvil con idéntico fabricante y modelo tendrán el mismo hash y por lo tanto serán consideradas (bajo nuestro criterio) como iguales.

El método hashCode

Es más rápido que el método equals. Cuando comparamos dos objetos…

1º - Si no tienen el mismo hashcode, son diferentes.

2º - Si tienen el mismo hashcode, se llamará al método equals que hará un análisis de igualdad más completo.



¿Pueden dos objetos tener el mismo hashCode y ser diferentes?

Por supuesto. Depende de cómo implementemos el método hashCode() en su clase:

```
@Override
public int hashCode() {
        return 1;
}
```

Con esta implementación todas las instancias de la clase tendrán el mismo hashCode().

Será equals la encargada de comprobar la igualdad con los criterios a la carta que establezcamos.

Resumen hashCode

Toda clase debe proveer de un método hashCode() que permita recuperar el hash asignado a cada instancia.

En el lenguaje de programación Java, un hash es un identificador de 32 bits que se almacena en un hash en la instancia de la clase.

Por defecto, si no lo sobreescribimos, utilizaremos la implementación de Object: el método devuelve por defecto un número entero basado en la dirección de memoria interna del objeto. Pero:

el hashCode no es la dirección de memoria interna de un objeto!!!

Objects.equals

El método static equals de la clase helper java.utils.Objects acepta dos objetos y concluye si ambos objetos son iguales o son diferentes:

1 - Si ambos objetos apuntan a null, entonces equals() devuelve true.

2 - Si alguno de los objetos apunta a null, entonces equals() devuelve false.

3 - Si ambos objetos son iguales, entonces equals() devuelve true.

Objects.equals

Vamos a echar un vistazo al método Objects.equals:

```
public static boolean equals(Object a, Object b) {
    return (a == b) || (a != null && a.equals(b));
  }
```

Como podemos ver el método Objects.equals evalúa por un lado si ambos objetos tienen la misma referencia con == en cuyo caso devuelve true. Por otro lado evalúa si se cumple que uno de los elementos no sea null y que a y b sean iguales mediante el método equals de la instancia de uno de los objetos.

La única diferencia apreciable entre utilizar este método equals static o el método equals de la instancia es el de no incurrir en NullPointerException en caso de que ambos objetos sean null.

El método equals de la clase String

Este método viene sobrecargado (@Override) en el JDK (java.Lang). Básicamente se ve así:

```
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    return (anObject instanceof String aString)
            && (!COMPACT_STRINGS || this.coder == aString.coder)
            && StringLatin1.equals(value, aString.value);
}
```

Lógicamente lo comparten todos los Strings y sirve para determinar si dos Strings son iguales.

Si le pasamos un null se producirá un NullPointerException.

El método hashCode de la clase String

Dos Strings con “textos” iguales deben tener el mismo hashCode, pero dos Strings con “textos” diferentes pueden tener el mismo hashCode.

Cuando dos Strings tienen el mismo código hash se produce un fenómeno conocido como hashcode collision. Hay muchos casos en los que la colisión de código hash se produce. Por ejemplo, "Aa" y "BB" tienen el mismo valor de código hash siendo cadenas diferentes.

No debemos confiar en el método hashCode() para comprobar si dos cadenas son iguales. La clase String sobrescribe esta función de la clase Object. Es usada por Java internamente cuando String es usado como clave de Map para las operaciones get() y put().