APRENDERAPROGRAMAR.COM

EL MÉTODO EQUALS EN JAVA. DIFERENCIA ENTRE IGUALDAD E IDENTIDAD. COMPARAR OBJETOS. EJEMPLOS. (CU00662B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

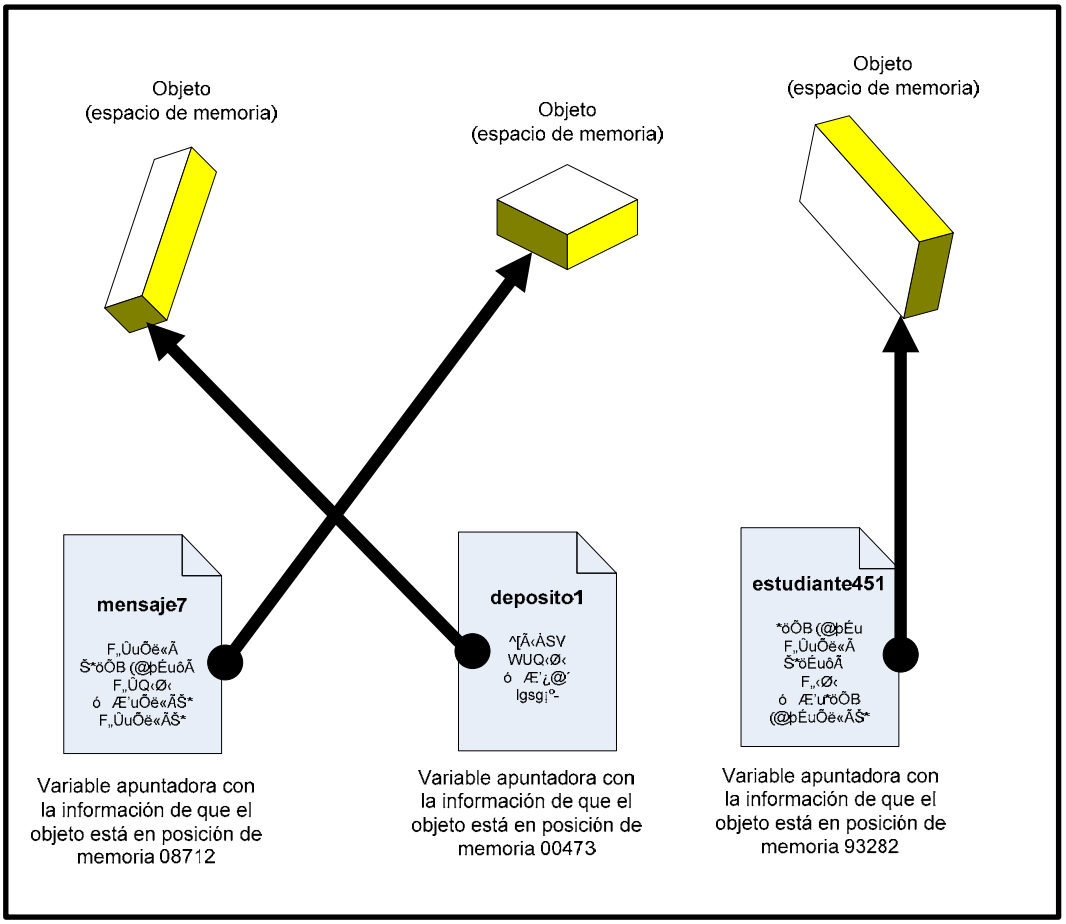
Resumen: Entrega nº62 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

EL MÉTODO EQUALS EN JAVA. DIFERENCIA ENTRE IGUALDAD E IDENTIDAD DE OBJETOS.

Recordemos ahora algo que hemos comentado anteriormente: **Un objeto es una cosa distinta a un tipo primitivo, aunque "porten" la misma información.** Vamos a tratar de profundizar y aclarar con más detenimiento la diferencia entre objetos y tipos primitivos. Para ello vamos a valernos de un símil: consideremos que un objeto es una caja que puede contener muchas cosas (por ejemplo, un caldero, un bolígrafo, un ventilador, una sábana, etc.).

La variable que referencia al objeto sería una nota de papel donde tenemos escrito dónde se encuentra la caja. Decimos que esta variable contiene una referencia o puntero al objeto.



En este ejemplo vemos cómo cuando usamos una instrucción como *deposito1 = new Deposito();*, en realidad **estamos creando dos cosas**: la variable apuntadora denominada deposito1, que sería como

nuestra nota de papel que nos dice dónde está el objeto, y el objeto o espacio de memoria que queda reservado. Para acceder al objeto hemos de usar la variable apuntadora. La memoria es como un gran almacén con miles de cajas apiladas y si no sabemos dónde está una caja no podremos encontrarla. Cuando creamos un tipo primitivo, creamos una sola cosa, pensemos por ejemplo que creamos un bolígrafo. Y ese bolígrafo llevaría una etiqueta con su nombre (nombre de la variable). Para gestionar tipos primitivos no tenemos la duplicidad entre apuntador y espacio de memoria. Tenemos un espacio de memoria al que podemos acceder directamente.

¿Por qué estas dos formas de gestionar la información? ¿Por qué no almacenar todo de forma que sea accesible directamente? La razón para ello estriba en la eficiencia de computación: todo está pensado para optimizar los rendimientos de los ordenadores y de los programas. Nosotros no tenemos por qué profundizar en estas cuestiones, pero sí conocerlas porque tienen relevancia de cara a la programación.

Consideremos que cuando hacemos una petición a la memoria, esta puede ser de dos tipos:

a) Le pedimos un tipo primitivo. Equivaldría a decir directamente “Dame el bolígrafo 783”. Y esta información se nos sirve.
b) Le pedimos un objeto. Equivaldría a decir “Dame la caja que está en la posición 93282”. Y esta información se nos sirve.

Supongamos que mensaje7 es una variable apuntadora a una caja que contiene 2 bolígrafos, y que estudiante451 es otra variable apuntadora a una caja que contiene otros 2 bolígrafos. Vamos a plantear algunas preguntas:

PREGUNTA	RESPUESTA	RAZONAMIENTO
¿Es idéntico mensaje7 que estudiante451? **mensaje7 == estudiante451**	No, no es igual. False	El contenido de mensaje7 es “El objeto está en posición de memoria 08712”, mientras que el contenido de estudiante451 es “El objeto está en posición de memoria 93282”. Por tanto no son iguales.
¿Es igual el contenido de mensaje7 que el contenido de estudiante451? **estudiante7.equals(estudiante451)**	Depende	Depende de qué consideremos ser igual. Si consideramos que por contener 2 bolígrafos el contenido es igual la respuesta es sí. Si consideramos que además hemos de comprobar que los bolígrafos sean de la misma marca y color, ahora mismo no sabríamos decir si los contenidos son iguales o no.

Llegamos a conclusiones importantes:

1) La comparación usando == no se debe usar para comparar el "contenido" de los objetos, sino únicamente para comparar la información de las variables apuntadoras. Usar == para comparar objetos es un error frecuente en programadores con poca experiencia.

2) Para comparar el contenido de los objetos hemos de usar, en general, un método especial del que disponen todos los objetos denominado **equals**. Equals se basa en una definición de igualdad que ha de estar establecida. Por ejemplo, podríamos definir que dos objetos son iguales si contienen el mismo número de bolígrafos, de la misma marca, modelo y color. Para algunos tipos de objetos, como los String, la definición de igualdad que viene incorporada en el API de Java nos va a resultar suficiente. Es decir, *if (miCadena1.equals (miCadena2))* va a funcionar correctamente. Sin embargo, si hemos definido un tipo Deposito y tratamos de utilizar *if (deposito1.equals (deposito2))* no obtendremos un resultado satisfactorio puesto que no hemos definido cuál es el criterio de igualdad.

Planteémonos si es correcta esta sintaxis: *if (entradaTeclado.substring(0,1)=="j")*. Primera cuestión: ¿Qué devuelve el método substring de la clase String? De acuerdo con la documentación del API de Java, devuelve un String (un objeto). Segunda cuestión: ¿Qué es "j"? Al estar entre comillas, se trata de un objeto String. Por tanto estamos preguntando que si un puntero es igual a "j". Esto no tiene un resultado previsible. Por tanto esta sintaxis no la podemos considerar correcta, ni siquiera aunque aparentemente pueda parecer que funcione. Lo correcto sería usar: *if (entradaTeclado.substring(0,1).equals("j"))*. Aquí lo que estamos haciendo es comparar el contenido de un objeto con el contenido de otro objeto.

Si usamos == comparando dos objetos, en realidad lo que haremos es preguntar si las dos variables que referencian al objeto tienen el mismo puntero, es decir, si ambos objetos son el mismo objeto (identidad).

La comparación de igualdad entre objetos requiere que se defina antes en base a qué se va a considerar que dos objetos son iguales. Por ejemplo, para un objeto String parece bastante claro, será el contenido de la cadena, pero para un objeto Persona quizás no esté tan claro: ¿bastará con que tengan igual DNI o habrá que tener en cuenta nombres y apellidos y otros datos para evitar que un error en el DNI haga que dos personas se consideren iguales? La definición de igualdad se hace con el método equals, que es habitual que se implemente para cualquier clase. Esto lo veremos más adelante. Ahora repasaremos únicamente cómo comparar dos cadenas. La sintaxis será:

Cadena1.equals(Cadena2) ó *("Texto1").equals("Texto2")* devuelve true si la Cadena1 / Texto1 y la Cadena2 / Texto2 tienen el mismo contenido, o false si su contenido es diferente. Por ejemplo: *System.out.println ("Coco y coco son iguales? " + ("Coco").equals("coco"));* devuelve false (cuentan mayúsculas, minúsculas y espacios).

Un método puede aplicarse sobre lo devuelto por otro método. Por ejemplo:

System.out.println ("Primera palabra de las dos primeras entradas iguales? " + entrada1.getPrimeraPalabra().equals(entrada2.getPrimeraPalabra()));

El uso del método equals es exclusivo para objetos. Recordemos que los tipos primitivos no tienen métodos. Por tanto, **los tipos primitivos los compararemos de la forma tradicional usando ==.**

EJERCICIO

Crea una clase en cuyo método main ejecutes una comparación letra a letra usando equals de dos palabras usando bucles. Por ejemplo si las palabras son “avispa” y “ave” el programa debe dar como resultado: ¿Letra 1 igual en las dos palabras? True. ¿Letra 2 igual en las dos palabras? True ¿Letra 3 igual en las dos palabras? False ¿Letra 4 igual en las dos palabras? La palabra 2 no tiene letra 4 ¿Letra 5 igual en las dos palabras? La palabra 2 no tiene letra 5 ¿Letra 6 igual en las dos palabras? La palabra 2 no tiene letra 6.

Puedes comprobar si tus respuestas son correctas consultando en los foros aprenderaprogramar.com.

Próxima entrega: CU00663B

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188