APRENDERAPROGRAMAR.COM

# LA PALABRA CLAVE THIS EN JAVA. CONTENIDO NULL POR DEFECTO DE UN OBJETO. SOBRECARGA DE NOMBRES. EJEMPLOS (CU00654B)

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

Fecha revisión: 2029

**Resumen:** Entrega nº54 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

## PALABRA CLAVE THIS EN JAVA. CONTENIDO NULL POR DEFECTO DE UN OBJETO.

Ya hemos visto en el epígrafe anterior que la palabra clave *this* puede ser usada para invocar a un constructor. Sin embargo, su uso quizás más frecuente en Java tiene lugar en otro contexto: cuando existe sobrecarga de nombres. La sobrecarga de nombres se da cuando tenemos una variable local de un método o constructor, o un parámetro formal de un método o constructor, con un nombre idéntico al de un campo de la clase.

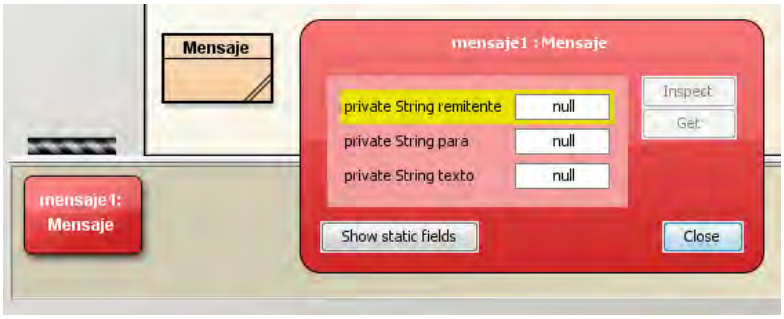

Este sería un mal ejemplo de sobrecarga de nombres, tanto con parámetros como con variables locales:

```
public class Mensaje {
   //Campos
   private String remitente;
   private String para;
   private String texto;

   //Constructor con sobrecarga de nombres al coincidir nombres de parámetros con los de campos
   public Mensaje (String remitente, String para, String texto) {
      remitente = remitente; //¿Cómo va a saber Java cuál es el parámetro y cuál el campo?
      para = para; //¿Cómo va a saber Java cuál es el parámetro y cuál el campo?
      texto = texto; //¿Cómo va a saber Java cuál es el parámetro y cuál el campo?
   } //Cierre del constructor

   //Método con sobrecarga de nombres al coincidir un parámetro con un campo
   public void setRemitente (String remitente) {
      remitente = remitente; //¿Cómo va a saber Java cuál es el parámetro y cuál el campo?
   } //Cierre del método

   //Método con sobrecarga de nombres al coincidir una variable local con un campo
   public void extraerFraccionTexto () {
      String texto = ""; //Esto supone declarar una variable local que "tapa" al campo
      texto = texto.substring (0, 5);  //¿Cómo va a saber Java si nos referimos al campo?
   } //Cierre del método

} // Cierre de la clase
```

Escribe y compila el código anterior. El código, efectivamente compila. ¿Por qué? Porque Java tiene previstos mecanismos para resolver conflictos de nombres y aplica una reglas. En concreto, la regla de que "un nombre hace referencia a la variable más local de entre las disponibles". Y el carácter de local se interpreta de la siguiente manera:

**Variable local > Parámetro formal > Campo de clase**

Crea ahora un objeto de tipo Mensaje e introduce un texto como *remitente*, por ejemplo "Juan", otro como *para*, por ejemplo "Pedro" y otro como *texto*, por ejemplo "Saludos desde Buenos Aires". A continuación, inspecciona el estado del objeto. El resultado será algo así:




Interpretemos ahora por qué nos aparece *null* como contenido de los atributos del objeto. En el constructor hemos definido tres parámetros: *remitente*, *para* y *texto*. Luego hemos indicado que *remitente = remitente;*. ¿Qué variable usa Java? Tiene que elegir entre usar el campo o usar el parámetro del método. No puede usar ambos porque no podría saber cuándo usar uno y cuándo usar otro. El conflicto lo resuelve utilizando el parámetro del método, es decir, interpretando que "el parámetro es igual al parámetro". Esto no tiene ningún efecto, lo que significa que el atributo *remitente* se queda sin inicializar. Lo hemos declarado, pero no lo hemos inicializado.

En nuestra clase tenemos tres campos o variables de instancia cuyo ámbito es toda la clase. El ámbito de los parámetros o variables locales es exclusivamente el constructor o método al cual se aplican.

Recordemos que un String es un objeto en Java. Un objeto no inicializado carece de contenido y esto nos lo informa Java indicándonos un contenido aparente *null*. La palabra clave *null* indica que un objeto se encuentra vacío, carente de contenido.

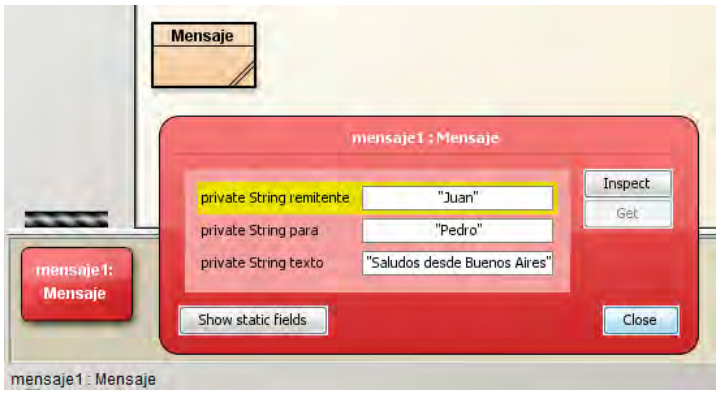
> ***La palabra clave null indica que un objeto se encuentra vacío, carente de contenido. Esto puede deberse a que no se ha inicializado, a que su contenido ha sido eliminado usando una instrucción del tipo nombreDelObjeto = null;, o a otros motivos. En el caso de un String, hay que diferenciar entre que el objeto tenga como contenido una cadena vacía, que al fin y al cabo es un contenido, o que carezca de contenido. La palabra clave null no es aplicable a los tipos primitivos, que no admiten una asignación del tipo nombreVariable = null. Un tipo primitivo no inicializado tendrá como contenido un valor en el rango de valores admisibles para el tipo como 0 para int o false para boolean. Recordar que por norma inicializaremos siempre cualquier objeto o variable de forma explícita.***

Una instrucción del tipo *if (remitente == null) { … }* nos puede servir para detectar si un objeto ha sido inicializado adecuadamente.

Volvamos ahora al código de nuestra clase Mensaje. El conflicto de nombres vamos a solventarlo haciendo uso de la palabra clave *this*. Escribiendo *this.nombreDelCampo* le indicaremos a Java que nos referimos al atributo de la clase en vez de a una variable local o parámetro. Veámoslo aplicado en el código:

```
public class Mensaje {
   private String remitente;
   private String para;
   private String texto;

   //Constructor con sobrecarga de nombres al coincidir nombres de parámetros con los de campos
   public Mensaje (String remitente, String para, String texto) {
      this.remitente = remitente; //this.remitente es el campo y remitente el parámetro
      this.para = para; //this.para es el campo y para el parámetro
      this.texto = texto; //this.texto es el campo y texto el parámetro
   }

   //Método con sobrecarga de nombres al coincidir un parámetro con un campo
   public void setRemitente (String remitente) {
      this.remitente = remitente; //this.remitente es el campo y remitente el parámetro
   }

   //Método con sobrecarga de nombres al coincidir una variable local con un campo
   public String extraerFraccionTexto () {
      String texto = ""; //texto es una variable local
      texto = this.texto.substring (0, 5);  //this.texto es el campo de los objetos de la clase
      return texto;
   }  }
```

Crea ahora un objeto de tipo Mensaje e inicialízalo introduciendo valores para los parámetros requeridos por el constructor. El resultado es que ahora sí se produce una inicialización correcta porque hemos definido adecuadamente cómo han de gestionarse los nombres de variables.




En resumen, **usando la palabra clave *this* podemos evitar que los parámetros o variables locales tapen a las variables globales.** Para interpretar el significado de *this*, podemos pensar que hace referencia al "objeto actual" en un momento dado. Repetir los nombres de campos como parámetros y como variables locales queda a elección de cada programador. En sí misma, esta práctica no puede considerarse ni buena ni mala siempre que se mantenga una coherencia y lógica global de nombres. La

realidad es que la repetición de nombres es muy frecuente en la práctica de la programación porque a la hora de desarrollar programas largos intervienen muchas variables y la búsqueda de nombres distintos puede hacer perder tiempo al programador y hacer más difícil de seguir el código.

**EJERCICIO**

Define una clase Profesor considerando los siguientes atributos de clase: nombre (String), apellidos (String), edad (int), casado (boolean), especialista (boolean). Define un constructor que reciba los parámetros necesarios para la inicialización y otro constructor que no reciba parámetros. El nombre de los parámetros debe ser el mismo que el de los atributos y usar this para asignar los parámetros recibidos a los campos del objeto. Crea los métodos para poder establecer y obtener los valores de los atributos, con sobrecarga de nombres y uso de this en los métodos setters (los que sirven para establecer el valor de los atributos). Compila el código para comprobar que no presenta errores, crea un objeto usando el constructor con sobrecarga. Comprueba que se inicializa correctamente consultando el valor de sus atributos después de haber creado el objeto. Usa los métodos setters y comprueba que funcionen correctamente.

Para comprobar si es correcta tu solución puedes consultar en los foros aprenderaprogramar.com.

**Próxima entrega:** CU00655B

**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