****

**Paradigma**

**Orientado a Objetos**

**Módulo 14:**

**Herencia de Constructores.**

**por Fernando Dodino**

**Versión 1.0**

**Enero 2018**

Distribuido bajo licencia [Creative Commons Share-a-like](https://creativecommons.org/licenses/by-sa/4.0/legalcode)

**Indice**

[1 Introducción](#_t0c6cgi8x2mv)

[1.1 Herencia básica de constructores](#_pfpf65o6x69t)

[1.2 La definición de constructores inhabilita la herencia](#_bd7l67229jre)

[2 Delegación de constructores con super](#_j9jzj1xu4no3)

[3 Delegación de constructores con self](#_tzrtbh45tz0n)

[4 Herencia de un WKO con constructores explícitos](#_magj4m3elpf6)

[5 Resumen](#_sanoxr9kny1w)

# 1 Introducción

El mecanismo de herencia permite que una clase tome definiciones de su superclase, que por el momento eran atributos y comportamiento. Los constructores también participan de este concepto, como veremos a continuación.1

## 1.1 Herencia básica de constructores

Dada esta jerarquía[[1]](#footnote-0)

**class** Pelota {

**var** color = "blanco"

**constructor**() { }

**constructor**(\_color) { color = \_color }

}

**class** PelotaFutbol **inherits** Pelota {

**var** duenio

}

Por defecto, **Wollok permite que PelotaFutbol herede los constructores de la superclase siempre que no defina ningún constructor**. Esto significa que es válido construir un objeto PelotaFutbol de las siguientes maneras:

**>>> new PelotaFutbol()**

**un/a PelotaFutbol[color=blanco, duenio=null]**

**>>> new PelotaFutbol("gris")**

**un/a PelotaFutbol[color=gris, duenio=null]**

De todas maneras no estamos inicializando todas las referencias correctamente, lo que nos puede llevar a errores derivados de este objeto inconsistente:

**>>> const pelotaGris = new PelotaFutbol("gris")**

**un/a PelotaFutbol[color=gris, duenio=null]**

**>>> pelotaGris.duenio().nombre()**

**wollok.lang.MessageNotUnderstoodException: un/a PelotaFutbol[color=gris, duenio=null] no entiende el mensaje duenio()**

**at wollok.lang.Object.messageNotUnderstood(messageName,parameters) (classpath:/wollok/lang.wlk:228)**

**>>>**

En ese caso tenemos dos formas de resolverlo:

* asignar una referencia a duenio
* generar un constructor propio para PelotaFutbol

## 1.2 La definición de constructores inhabilita la herencia

Vamos a generar entonces un constructor específico para PelotaFutbol.

**class** Pelota {

**var** color = "blanco"

**constructor**() { }

**constructor**(\_color) { color = \_color }

}

**class** PelotaFutbol **inherits** Pelota {

**var** duenio

**constructor**(\_color, \_duenio) {

color = \_color

duenio = \_duenio

}

}

Como consecuencia de esa definición Wollok inhibe los constructores que la jerarquía de superclases tenga. Por lo tanto, la única forma correcta de construir una pelota de fútbol es pasándole el color y el dueño:

**>>> new PelotaFutbol()**

**ERROR: Número de argumentos incorrecto. Debe ser new PelotaFutbol(\_color,\_duenio) (linea: 1)**

**>>> new PelotaFutbol("blanco")**

**ERROR: Número de argumentos incorrecto. Debe ser new PelotaFutbol(\_color,\_duenio) (linea: 1)**

**>>> new PelotaFutbol("blanco", pepe)**

**un/a PelotaFutbol[color=blanco, duenio=Pepe]**

# 2 Delegación de constructores con super

Quizás el lector haya advertido en el ejemplo anterior una duplicidad de ideas:

**class** Pelota {

**var** color = "blanco"

**constructor**() { }

**constructor**(\_color) { color = \_color }

}

**class** PelotaFutbol **inherits** Pelota {

**var** duenio

**constructor**(\_color, \_duenio) {

color = \_color

duenio = \_duenio

}

}

En este caso, solo se trata de una asignación directa. Pero si recibiéramos el color y en función de ese color determináramos el precio, o si estuvieran involucradas más tareas, sería bueno no repetir esa línea en la superclase y en todas las subclases que lo necesiten. Para tal fin vamos a delegar el constructor al constructor de la superclase, de la siguiente manera:

**class** Pelota {

**var** color = "blanco"

**constructor**() { }

**constructor**(\_color) { color = \_color }

}

**class** PelotaFutbol **inherits** Pelota {

**var** duenio

**constructor**(\_color, \_duenio) = **super**(\_color) {

duenio = \_duenio

}

}

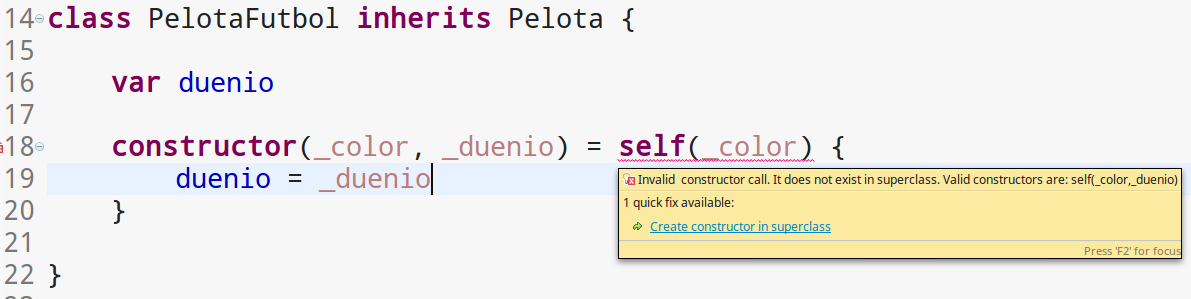
Cuando en la consola instanciemos una pelota de fútbol:

**>>> new PelotaFutbol("blanco", pepe)**

Quien responde es el único constructor PelotaFutbol que a su vez delega en el constructor de un solo parámetro que está en la superclase. Aquí debemos usar super ya que escribir

**constructor**(\_color, \_duenio) = **self**(\_color) {

produce un error ya que no existe un constructor definido en PelotaFutbol.



# 3 Delegación de constructores con self

Una pelota de basket sabe

* su fecha de confección
* y nos interesa saber si tiene dueño

Supongamos que tenemos dos constructores:

* uno donde ingresamos el color (la mínima definición posible)
* y otro donde pasamos la referencia al color y al dueño

**class** PelotaBasket **inherits** Pelota {

**const** fechaConfeccion

**const** tieneDuenio

**constructor**(\_color, \_duenio) = **self**(\_color) {

tieneDuenio = \_duenio != **null**

}

**constructor**(\_color) = **super**(\_color) {

fechaConfeccion = **new** Date()

tieneDuenio = **false**

}

**method** esNueva() = **new** Date() - fechaConfeccion < 30

}

En este caso quisimos inicializar en el constructor de un solo parámetro las dos referencias, entonces:

1. delegamos al constructor de la superclase (para no repetir la inicialización de la referencia color)
2. agregamos nuestras propias definiciones (en realidad no son del todo necesarias ya que podría hacerse directamente en la declaración de las referencias const, pero sirve para mostrar la variante con fines didácticos)
3. el constructor de dos parámetros que recibe dueño y color, delega al constructor de un solo parámetro, para evitar la repetición de ideas. Y además cambia la definición de la referencia tieneDuenio en base al dueño que recibe.

Ahora sí podemos crear una pelota de básket de dos maneras distintas:

**>>> new PelotaBasket("naranja")**

**a PelotaBasket[color=naranja, fechaConfeccion=a Date[day = 3, month = 2, year = 2018], tieneDuenio=false]**

**>>> new PelotaBasket("naranja", "david")**

**a PelotaBasket[color=naranja, fechaConfeccion=a Date[day = 3, month = 2, year = 2018], tieneDuenio=true]**

**>>>**

# 4 Herencia de un WKO con constructores explícitos

Recordemos el ejemplo de las aves, donde

* existen muchas aves, todas vuelan y comen de la misma manera
* pepita y chichita son aves que tienen un comportamiento específico y las quiero modelar como objetos conocidos (wko)

**class** Ave {

**var** energia = 100

**method** energia() = energia

**method** volar(kms) { energia -= kms + 10 }

**method** comer(gramos) { energia += 4 \* gramos }

}

**object** pepita **inherits** Ave { }

**object** chichita **inherits** Ave {

**override** **method** comer(gramos) {

energia = energia \* gramos

}

}

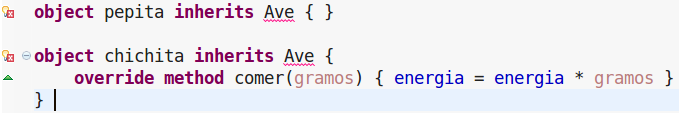
Luego de conversar con el usuario, nos comenta que cada ave debería inicializarse con una energía diferente. Entonces deberíamos generar un constructor que defina la energía inicial de cualquier ave...

**class** Ave {

**var** energia

**constructor**(\_energia) { energia = \_energia }

Pero ¿qué pasa con pepita y chichita? No tienen constructor, entonces dejaron de compilar:



Para resolver el problema, vamos a escribir la definición de pepita y chichita indicando cuál es la energía inicial que deben tener[[2]](#footnote-1):

**object** pepita **inherits** Ave(100) { }

**object** chichita **inherits** Ave(50) {

**override** **method** comer(gramos) { ... }

}

Ahora sí, pepita y chichita están correctamente definidas y podemos enviarle mensajes en un REPL:

**>>> pepita**

**pepita[energia=100]**

**>>> chichita**

**chichita[energia=50]**

# 5 Resumen

En este capítulo hemos conocido cómo funciona el mecanismo de herencia de los constructores en Wollok. Los constructores pueden heredarse evitando así duplicar ideas, o bien podemos crear constructores específicos basados en los de la superclase. Por último, si un wko hereda de una clase que tiene constructores específicos, debe definir los valores iniciales para que quede correctamente construido.

1. El ejemplo completo puede descargarse en <https://github.com/wollok/herencia-constructores> [↑](#footnote-ref-0)
2. El ejemplo completo puede descargarse en <https://github.com/wollok/herencia-pepita-WKO> [↑](#footnote-ref-1)