Paradigma
Orientado a Objetos

Módulo 07: Propiedades

por Fernando Dodino
Versión 1.0
Enero 2018

Indice

Nota: este feature estará disponible a partir del release 1.6.4

1 Repaso de accessors

Hemos visto en el módulo 2 que para poder acceder a la referencia de un objeto, éste debe definir accessors, en particular los getters y setters:

```
object pepita {
	var energia = 0

	method energia() = energia	// GETTER

	method energia(_energia) {	// SETTER
		energia = _energia
	}
	method volar() {
		energia = energia + 10
	}

}
```

De esa manera, podemos desde otra entidad (otro objeto, un test o un programa) conocer el valor de la referencia energia para pepita:

```
import pepita.*

describe "tests para pepita" {

	test "energia inicial para pepita" {
		assert.equals(100, pepita.energia())
	}

	test "energia para pepita luego de volar" {
		pepita.volar()
		assert.equals(90, pepita.energia())
	}
}
```

En el ejemplo recién contado, tomamos la decisión de publicar el getter y el setter, aunque en realidad solo estamos utilizando el getter en el test. De hecho, podríamos cuestionar si definir el setter es una decisión de diseño correcta, ya que la energía se va modificando cada vez que pepita vuela (y eventualmente cuando coma).

2 Código boilerplate

Pensemos en la definición de un auto cualquiera:

```
object dodge1500 {
	var color
	var kilometraje
	var conAire = false
	const patente = "RVM 363"
	const anioPatentamiento = 1979
}
```

Para poder definir los atributos, tuvimos que pensar qué referencias se pueden modificar y cuáles no: por eso la patente y el año son inmutables, mientras que permitiremos ponerle aire al auto, o cambiarle el color. Por supuesto el kilometraje se incrementará a medida que el auto vaya rodando.

La segunda decisión que acompaña a los atributos, es qué tipo de *accessors* definiremos para cada uno:

- para las referencias const vamos a publicar solo los getters
- en el caso del aire y el color, podemos publicar tanto getters como setters
- y en el caso del kilometraje, podríamos publicar un setter, pero vamos a preferir escribir un getter y tener un método de negocio realizarViaje(kilometros) para que incremente el kilometraje.

```
object dodge1500 {
	var color
	var kilometraje
	var conAire = false
	const patente = "RVM 363"
	const anioPatentamiento = 1979

	method patente() = patente
	method anioPatentamiento() = anioPatentamiento
	method conAire() = conAire
	method conAire(_conAire) { conAire = _conAire }
	method color() = color
	method color(_color) { color = _color }
	method kilometraje() = kilometraje
	method realizarViaje(kilometros) {
		kilometraje = kilometraje + kilometros
```

```
	}
}
```

La definición está llena de ***código boilerplate***, burocracia que no solo es tediosa de escribir, sino que también se entremezcla con el código que verdaderamente nos importa, el que está asociado a decisiones del negocio.

3 Propiedades en Wollok

3.1 Definición

Una propiedad en Wollok permite explicitar un contrato específico para una referencia:

- si la referencia es variable (var), Wollok genera su getter y setter
- si la referencia es constante (const), Wollok genera el getter

La creación de los accessors es **implícito**, no se visualiza en el código ya que la idea es conservar únicamente el comportamiento definido por el negocio.

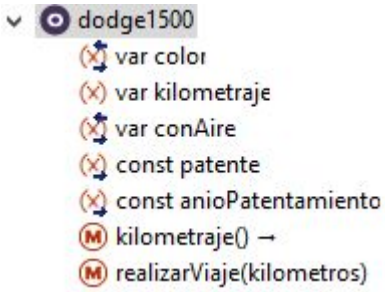
El mismo ejemplo de arriba quedaría así:

```
object dodge1500 {
	var property color
	var kilometraje
	var property conAire = false
	const property patente = "RVM 363"
	const property anioPatentamiento = 1979

	method kilometraje() = kilometraje

	method realizarViaje(kilometros) {
		kilometraje = kilometraje + kilometros
	}
}
```

Como no queremos publicar un setter para el atributo kilometraje, elegimos

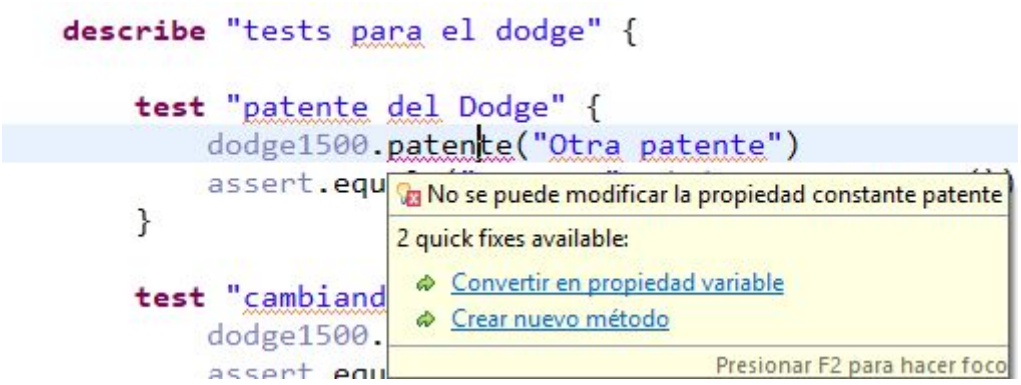
- no aplicar el modificador property para kilometraje
- y definir manualmente un getter específico para kilometraje

En la vista Esquema (Outline) se visualizan los atributos y propiedades const (una sola flecha saliente) y var (flechas entrante y saliente):



3.2 La propiedad constante solo define getter

Recordemos que si la patente es una propiedad cuya referencia es constante, no tendremos setters:



3.3 Redefiniendo accessors implícitos

En el caso de definir un método accessor propio, éste pisa el implícito que define la property, tanto para getters como para setters. Continuando con el ejemplo del auto

```
object dodge1500 {
	...
	const property anioPatentamiento = 1979

	method anioPatentamiento() = anioPatentamiento + 1
```

Si construimos un test específico para el año, veremos que se toma en cuenta la definición explícita del método anioPatentamiento() escrita por nosotros:

```
describe "tests para el dodge" {

	test "anio patentamiento de un auto común" {
		assert.equals(1980, dodge1500.anioPatentamiento())
```

```
}
```

4 Resumen

Wollok permite la definición de propiedades, lo que evita al programador tener que escribir o leer el código de los accessors (getters y/o setters) cuya necesidad es meramente administrativa.