Paradigma
Orientado a Objetos

Módulo 03:
Introducción a Polimorfismo.

por Fernando Dodino
Versión 2.1
Agosto 2017

Indice

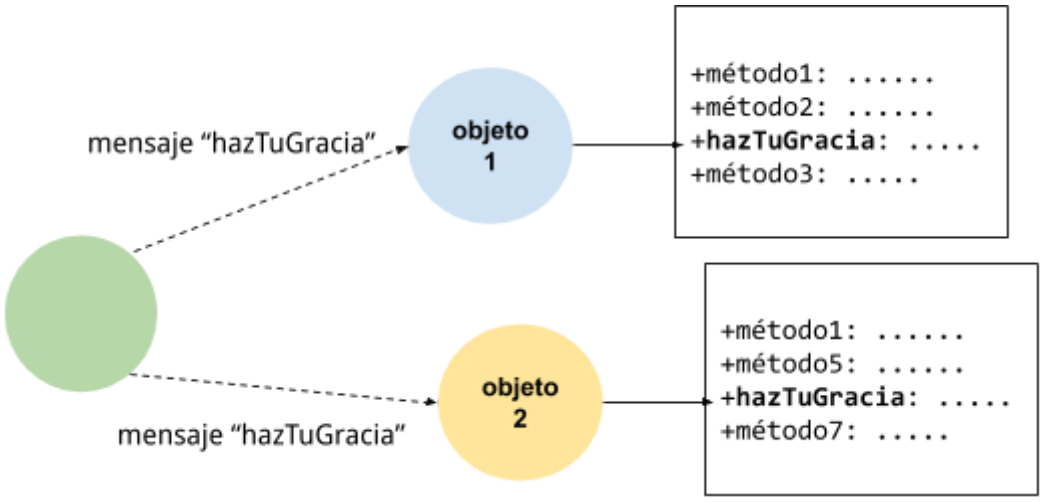
1 Introducción al Polimorfismo

¿Qué pasa si dos objetos son capaces de responder a un mismo mensaje?
El que manda el mensaje puede cambiar la referencia de un objeto a otro sin notar la diferencia.

Se produce entonces un concepto fundamental en el paradigma, cuando tenemos 3 objetos

- un objeto que envía un mensaje
- y dos objetos que implementan el mensaje de dos maneras diferentes (cada uno define un método distinto)

El observador puede cambiar la referencia sin preocuparse por el envío del mensaje, que permanece sin cambios. Decimos entonces que los dos objetos que comparten cierta interfaz en común son **polimórficos** para ese observador.



polimorfismo: dos objetos son polimórficos a la vista de un tercero porque puede enviar el mismo mensaje sin importar cómo estén implementados los métodos

Ojo, no es necesario que los dos objetos polimórficos definan exactamente los mismos métodos. Alcanza con que implementen al menos un mensaje en común (como en el ejemplo el método hazTuGracia).

1.1 Analogía

Para ilustrar este concepto utilizaremos algunas analogías, veamos el botón *Play* en cada uno de estos artefactos...


AT&T
9:42 AM
Done
00:00
-01:28
YouTube


SAMSUNG
VCR & DVD PLAYER
DVD
VIDEO
OPEN/CLOSE
VHS
DVD-V9650
EJECT
POWER
DOLBY DIGITAL
DivX
12:48:36
CHANNEL
DVD/VCR
HD
REC


PAUSE
REW
FF
PLAY
STOP


AUTO STOP
COUNTER
SAVE
commodore
RECORD
PLAY
REWIND
F. FWD
STOP
EJECT

Todos los artefactos entienden el mensaje ▶ . Es decir que si lo lleváramos a una representación en objetos, podríamos enviarles a todos el mismo mensaje:

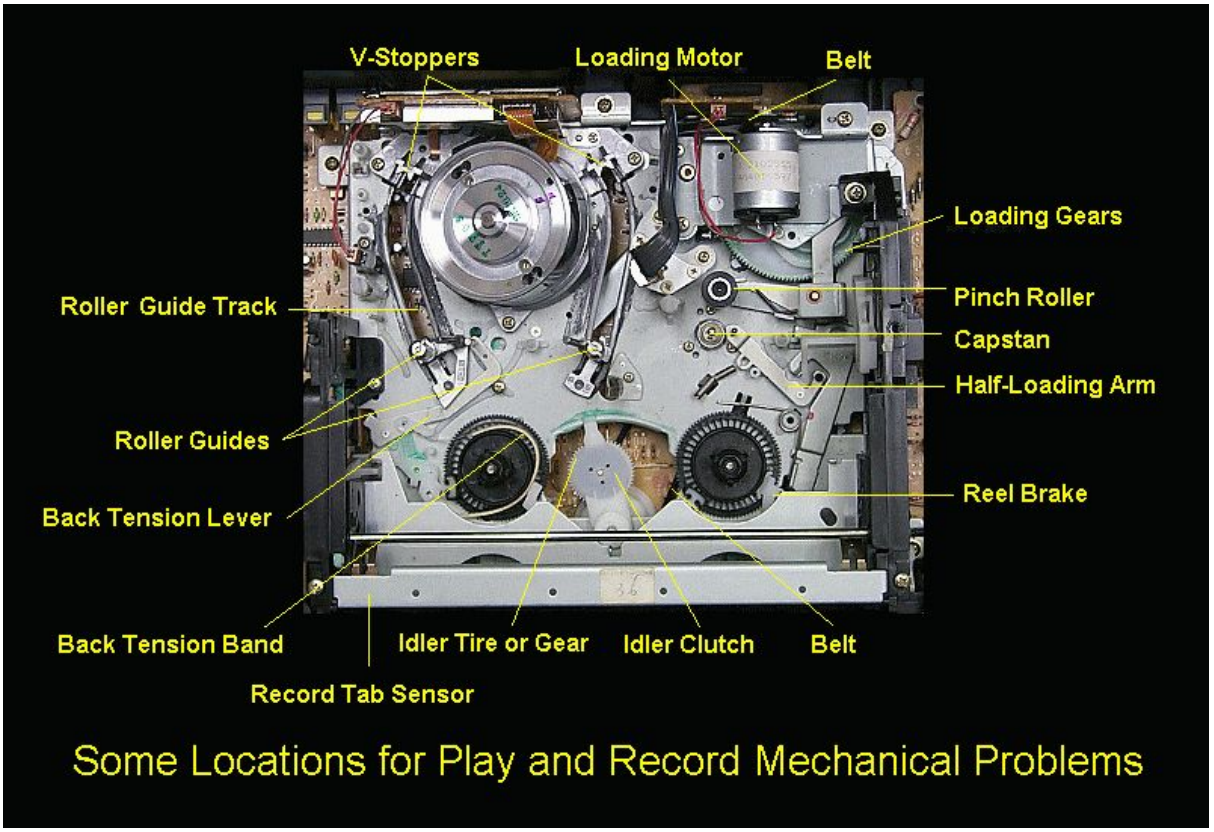
```
Wollok interactive console (type "quit" to quit):
>>> ipod.play()
>>> dvdLiving.play()
>>> vhsRecorder.play()
```

No obstante, está claro que cada artefacto hace cosas diferentes, porque sus implementaciones son diferentes (los métodos no son iguales). Por otra parte, no podemos pretender que funcionen igual, y no lo harán: un video reproductor no puede grabar, de la misma manera que no podemos ir a la pista 8 de un cassette. Pero sí es útil que varios objetos compartan un mensaje, para que alguien use a esos objetos dentro de algún contexto.

> ¿Para quién es bueno eso? No para los objetos que son polimórficos, sino **para quienes usamos esos artefactos**.

1.2 Repaso mensaje y método

¿Por qué no quiero ver cómo se implementa el método en cada objeto?



Ah, cierto. La complejidad.

1.3 Breve ejemplo

Tenemos dos empleados de un negocio que vende empanadas, Galván y Baigorria. Galván cobra un sueldo fijo (hoy es 15.000, pero eso puede modificarse). Baigorria cobra 15 pesos por cada empanada que vende. Generamos el ejemplo en Wollok[1]:

```
object galvan {
	var sueldo = 15000

	method sueldo() = sueldo

	method sueldo(_sueldo) { sueldo = _sueldo }
}

object baigorria {
	var cantidadEmpanadasVendidas = 100
	var montoPorEmpanada = 15

	method venderEmpanada() {
		cantidadEmpanadasVendidas = cantidadEmpanadasVendidas + 1
	}

	method sueldo() = cantidadEmpanadasVendidas * montoPorEmpanada
}
```

Lo probamos en un REPL:

```
>>> baigorria.sueldo()
1500
>>> baigorria.venderEmpanada()
>>> baigorria.sueldo()
1515
>>> galvan.sueldo()
15000
```

Vemos que tanto a baigorria como a galvan podemos preguntarle el sueldo. No nos importa cómo lo resuelve cada uno, pero sí que en algún punto comparten la misma interfaz: mensaje sueldo() sin parámetros, a baigorria no le podemos asignar el sueldo.

```
>>> baigorria.sueldo(29)
ERROR: baigorria does not understand sueldo(29) (line: 1)
```

[1] El ejemplo puede descargarse de https://github.com/wollok/polimorfismo-empanadasGimenez

Vamos a agregar entonces un tercer objeto, un observador para estos... tenemos un empleador, justamente el señor Giménez, que es quién le paga el sueldo a estos empleados. Necesitamos poder decirle a Giménez que le pague a Baigorria o a Galván. Cuando paga, disminuye su dinero. Vamos a dejar de lado la recepción del dinero por parte de los empleados, ya que extendería el ejemplo didáctico.

Agregamos entonces el objeto que representa al empleador, con su correspondiente método para pagar a un empleado. Asumimos que su "capital" o dinero inicial es de $ 300.000.

```
object gimenez {
   var dinero = 300000

   method dinero() = dinero
   method pagarA(empleado) { dinero = dinero - empleado.sueldo() }
}
```

¿Quién recibe el mensaje sueldo enviado por gimenez? Y... puede ser baigorria o galvan, viendo sólamente el código no podemos saber.

Por ejemplo, si en la consola ingresamos:

```
>>> gimenez.pagarA(baigorria)
>>> gimenez.dinero()
298485
```

Al evaluar la consulta, el parámetro de pagarA(empleado) pasa a ser baigorria, y por lo tanto empleado.sueldo() es un mensaje que termina provocando la evaluación del método sueldo() de baigorria. Pero bien podríamos haber hecho la consulta para galvan. El mensaje que manda el observador es siempre *empleado.sueldo( )*, pero el método que finalmente se termina evaluando, es decir, sueldo() del objeto baigorria o sueldo() del objeto gimenez, se determina en tiempo de ejecución.

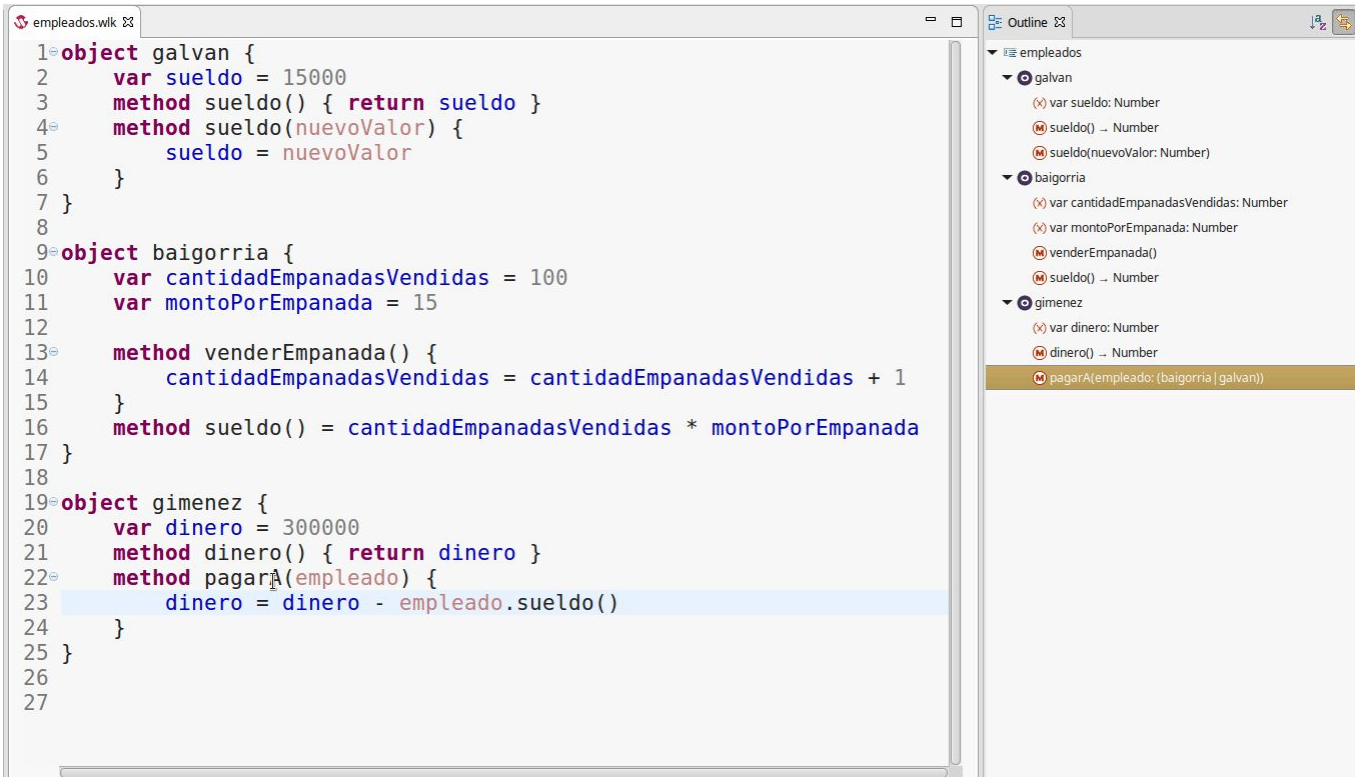
1.4 Polimorfismo y Tipos

Volvamos al método del objeto gimenez:

```
method pagarA(empleado) { dinero = dinero - empleado.sueldo() }
```

¿de qué tipo es el parámetro empleado? en principio, de cualquier objeto que entienda el mensaje sueldo() sin parámetros y que devuelva un número. El sistema de tipos de Wollok sabe que son por el momento dos los objetos posibles: baigorria y galvan.

Incluso, si generamos un nuevo objeto gutierrez que tenga implementado un método sueldo(), tan pronto grabemos el archivo el sistema de tipos lo detectará y actualizará los posibles tipos para el parámetro empleado en el método pagar:



2 Resumen

En este capítulo hemos conocido uno de los conceptos fundamentales del paradigma: el polimorfismo como una herramienta para agrupar objetos que tienen diferente comportamiento (distinto código) pero la misma interfaz. Entonces puedo enviarles mensajes sin saber cómo lo implementan, e incluso sin saber a qué objeto particular le estoy hablando: puedo cambiar la referencia sin tener que hacer cambios.