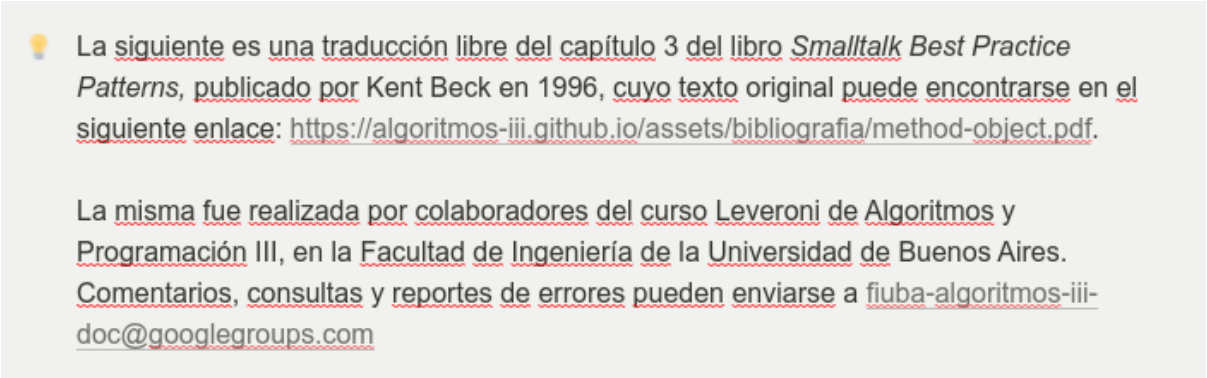
Method Object

Tienes un método que no se simplifica bien con Composed Method (pág. 21).



- ¿Cómo se escribe un método en el que muchas líneas de código comparten muchos argumentos y variables temporales?

El comportamiento en el centro de un sistema complejo suele ser complicado. Esa complejidad generalmente no se reconoce en un principio, por lo que el comportamiento se representa con un único método. Gradualmente ese método crece y crece, ganando más líneas, más parámetros y más variables temporales, hasta que es un desastre monstruoso.

Lejos de mejorar la comunicación, la aplicación de *Composed Method* a tal método sólo oscurece la situación. Dado que todas las partes de un método de este tipo generalmente necesitan todas las variables y parámetros temporales, cualquier parte del método que separes requiere seis u ocho parámetros.

La solución es crear un objeto para representar una invocación del método y utilizar el espacio de nombres compartido de las variables de instancia en el objeto para permitir luego una mayor simplificación mediante *Composed Method*. Sin embargo, estos objetos tienen un sabor muy diferente al de la mayoría de los objetos. La mayoría de los objetos son sustantivos, estos son verbos. La mayoría de los objetos son fácilmente explicables a los clientes, pero estos no lo son ya que no tienen un análogo en el mundo real. Sin embargo, los *Method Objects* valen su naturaleza extraña. Debido a que representan una parte tan importante del comportamiento del sistema, a menudo terminan en el centro de la arquitectura.

- Cree una clase nombrada a partir del método. Añada una variable de instancia para el receptor del método original, cada argumento y cada variable temporal. Añada un *Constructor Method* que tome el receptor original y los argumentos del método. Añada un método de instancia, *#computar*, implementado copiando el cuerpo del método original. Reemplace el método con uno que cree una instancia de la nueva clase y le envíe *#computar*.

Este es el último patrón que agregué a este libro. No lo iba a incluir porque lo uso muy pocas veces. Luego convenció a un cliente importante para que me diera un gran

contrato. Me di cuenta de que cuando lo necesitas, REALMENTE lo necesitas. El código se ve así:

```
Obligacion >> enviarTarea: unaTarea trabajo: unTrabajo
        | noProcesado procesado copiado ejecutado |
                ...150 lineas de codigo muy comentado...
```

Primero, probé *Composed Method*. Cada vez que intenté separar una parte del método, me di cuenta de que tendría que enviarle ambos parámetros y las cuatro variables temporales:

```
Obligacion >> prepararTarea: unaTarea
        trabajo: unTrabajo
        noProcesado: coleccionNoProcesada
        procesado: coleccionProcesada
        copiado: coleccionCopiada
        ejecutado: coleccionEjecutada
```

No solo esto era horrible, sino que la invocación resultante no ahorró ninguna línea de código (consulte *Indented Control Flow*, a continuación). Después de quince minutos de lucha, volví al método original y utilicé *Method Object*. Primero creé la clase:

```
Class: EnviadorDeTareas
        superclass: Object
        instance variables: obligacion tarea trabajo sinProcesar procesado copiado ejecutado
```

Se debe notar que el nombre de la clase es tomado directamente del selector del método original. También se debe notar que el receptor original, ambos argumentos y las cuatro variables temporales se convirtieron en variables de instancias.

El *Constructor Method* tomó el receptor original y ambos argumentos como parámetros:

```
EnviadorDeTareas class>>obligacion: unaObligacion tarea: unaTarea trabajo: unTrabajo
	^self new
		definirObligacion: unaObligacion
		tarea: unaTarea
		trabajo: unTrabajo
```

A continuación copié el código del método original. El único cambio que realicé fue textualmente reemplazar “unaTarea” con “tarea” y “unTrabajo” con “trabajo”, ya que los parámetros se nombran de manera diferente a las variables de instancia. Ah, y también elimine la declaración de las variables temporales, ya que ahora son variables de instancia.

```
EnviadorDeTareas>>computar
	… 150 lineas de codigo muy comentado ...
```

Luego cambie el método original para que cree e invoque un EnviadorDeTareas:

```
Obligacion>>enviarTarea: unaTarea trabajo: unTrabajo
	(EnviadorDeTareas
		obligacion: self
		tarea: unaTarea
		trabajo: unTrabajo) computar
```

Probé el método para asegurar que no había roto nada. Como lo único que estaba haciendo era mover texto de un lado para el otro, y lo hice con cuidado, el método modificado y su objeto asociado funcionaron en el primer intento.

Ahora siguió la parte divertida. Como todas las piezas del método ahora comparten las mismas variables de instancia puedo usar el Composed Method sin tener que pasar ningún parámetro. Por ejemplo, el bloque de código que preparaba una tarea se convirtió en un método llamado #prepararTarea

El trabajo entero tomó alrededor de dos horas, pero para cuando había terminado el método #computar se leía como documentación; había eliminado tres de las variables de instancia, el código tenía la mitad del largo original y había encontrado, y arreglado, un bug en el código original.