El Jefe Cochise y el acoplamiento de código

Gorka Urrutia

Version 1.0, 09/04/2018, phpsensei.es

Table of Contents

Sobre este documento

Consigue la última versión de este documento en https://phpsensei.es.

¡Cuidado con los rostros pálidos!

Eres el *Jefe Cochise*. Hoy es un alegre, tranquilo y suave día de primavera. Cabalgas por la pradera junto a tus más bravos y leales guerreros. La vida a veces es buena.

De pronto *Corazón Bravo* grita y señala hacia el horizonte. A lo lejos, una nube de polvo. Te acercas y descubres a un destacamento de salvajes y malencarados hombres blancos armados hasta los dientes. ¡Y se dirigen hacia el campamento! ¡Rápido! ¡Hay que avisar a la tribu!

Piensas que la mejor forma de avisar es a través de señales de humo. A todo correr, sacas tu portátil y te pones a teclear (¿he mencionado que es un universo paralelo?). Empiezas a crear una nueva clase para enviar la señal de humo pero recuerdas que *Perro Loco* había creado ya una clase para eso:

```
class SenalesHumo
{
    public $mensaje;

    public function enviar()
    {
        if ($this->mensaje) {
            echo "Mensaje enviado con señal de humo: " . $this->mensaje . PHP_EOL;
        }
    }
}
```

Así que la descargas usando Composer. Ya solo queda usarla en tu proyecto rápidamente. Como buen Apache que eres sabes que no debes modificar esta clase porque así las modificaciones que haga *Perro Loco* no sobreescribirán tus cambios. Entonces, qué mejor que crear una nueva clase llamada *MensajeroVeloz* para usar la clase *SenalesHumo*:

```php
class MensajeroVeloz
{
    private $mensaje;

    public function __construct($mensaje)
    {
        $this->mensaje = $mensaje;
    }

    public function enviarMensaje()
    {
        $senal = new SenalesHumo();
        $senal->mensaje = $this->mensaje;
        $senal->enviar();
    }
}

$mensajero = new MensajeroVeloz("¡Alerta! ¡Se acercan enemigos!");
$mensajero->enviarMensaje();
```

Pero, espera, hay que cifrar el mensaje para que no lo intercepte el enemigo. Decides que lo mejor es añadir el método *cifrarMensaje()* que encriptará el mensaje usando la clave pública de la tribu. Como eres un jefe sabio sabes que debes crear la propiedad *$mensaje* para no necesitar parámetros en los métodos:

```php
class MensajeroVeloz
{
    private $mensaje;

    public function __construct($mensaje)
    {
        $this->mensaje = $mensaje;
    }

    public function enviarMensaje()
    {
        $senal = new SenalesHumo();
        $senal->mensaje = $this->mensajeCifrado();
        $senal->enviar();
    }

    private function mensajeCifrado()
    {
        // Trampa, aquí debería ir el código para cifrar el mensaje
        return $this->mensaje;
    }
}
```

Todo va bien. Ha llegado el momento de ejecutar el programa y avisar a la tribu. ¡No! ¡Espera! Si envías las señales de humo el enemigo las verá y conocerá tu posición.

¡Maldición!

Caballo Loco te sugiere que uses una paloma mensajera. ¡Qué buena idea! Por suerte *Perro Loco* también escribió una clase para eso:

```
class PalomaMensajera
{
    public $mensaje;

    public function enviar()
    {
        if ($this->mensaje) {
            // Código para enviar la paloma mensajera
        }
    }
}
```

Como eres un hombre sabio decides que es mejor dejar el código de las señales de humo por si lo necesitas en otra ocasión. Así tu clase funcionará para cualquiera de los dos métodos de envío:

```
class MensajeroVeloz
{
  ...

  public function enviarMensaje($sistema)
  {
      if ($sistema=='senal-humo') {
          $mensajero = new SenalesHumo();
      }
      else if ($sistema=='paloma') {
          $mensajero = new PalomaMensajera();
      }
      else {
          return;
      }
      $mensajero->mensaje = $this->mensajeCifrado();
      $mensajero->enviar();
  }

  ...
}
```

Menos mal que *Perro Loco* ha usado las dos veces el método *enviar()* y así el código queda más limpio, ¡que los espíritus de nuestros ancestros le bendigan!

Las cosas empeoran

Al ir a buscar la paloma para enviarla descubres que *Bailando Con Lobos* se la ha regalado su prometida (¿A quién se le ocurrió acoger a este blanco en la tribu?).

Menos mal que *Hackeador Solitario* descubrió cómo colarse en la infernal red del telégrafo de los rostros pálidos. Y por suerte hay un poste cerca.

Miras en Packagist y encuentras un paquete que te puede servir:

```
class Telegrama
{
    public function enviarTelegrama($mensaje)
    {
      echo "Mensaje enviado por telegrama: " . $this->mensaje . PHP_EOL;
    }
}
```

¡Vaya! Esta clase es algo diferente. Habrá que modificar a *MensajeroVeloz*:

```
class MensajeroVeloz
{
  ...

  public function enviarMensaje($sistema)
  {

      if ($sistema=='senal-humo') {
          $mensajero = new SenalesHumo();
          $mensajero->mensaje = $this->mensajeCifrado();
          $mensajero->enviar();
      }
      else if ($sistema=='paloma') {
          $mensajero = new PalomaMensajera();
          $mensajero->mensaje = $this->mensajeCifrado();
          $mensajero->enviar();
      }
      else if ($sistema=='telegrama') {
          $mensajero = new Telegrama();
          $mensajero->enviarTelegrama($this->mensajeCifrado());
      }
      else {
          return;
      }
  }

  ...
}
```

Esta clase está teniendo cada vez peor pinta. Por culpa de los nervios tu vejiga está a punto de estallar y, mientras vas a aliviarte, alguien coge el portátil y teclea el fatídico comando:

```
$ composer update
```

¡Se ha actualizado el paquete *PalomaMensajera*! ¡Y ha cambiado! ¿Por qué *Perro Loco*? ¿Por qué?

Resulta que ahora *PalomaMensajera* tiene ahora este aspecto:

```
class PalomaMensajera
{
    private $mensaje;

    public function __construct($mensaje)
    {
        $this->mensaje = $mensaje;
    }

    public function enviar()
    {
        if ($this->mensaje) {
            echo "Mensaje enviado con paloma mensajera: " . $this->mensaje . PHP_EOL;
        }
    }
}
```

Perro Loco ha querido mejorar la clase haciendo la propiedad *$mensaje* privada. Y, en realidad, es una mejora. Pero para tí, *Jefe Cochise*, es una desgracia. Ahora te toca volver a cambiar la clase *MensajeroVeloz*.

Ya que hay que cambiar esta clase *Mirada Simplona* sugiere mejorarla con un switch:

```
class MensajeroVeloz
{
  public function enviarMensaje($sistema)
  {
      switch ($sistema) {
          case 'senal-humo':
              $mensajero = new SenalesHumo();
              $mensajero->mensaje = $this->mensajeCifrado();
              $mensajero->enviar();
              break;
          case 'paloma':
              $mensajero = new PalomaMensajera($this->mensajeCifrado());
              $mensajero->enviar();
              break;
          case 'telegrama':
              $mensajero = new Telegrama();
              $mensajero->enviarTelegrama($this->mensajeCifrado());
              break;
      }
  }
}
```

Un momento, un momento, no era ése el único cambio. *Perro Loco* también ha cambiado la clase *SenalesHumo*. Ha decidido que es mejor que la propiedad se llame *$senal* en vez de *$mensaje*:

```
class SenalesHumo
{
    public $senal;

    public function enviar()
    {
        if ($this->senal) {
            echo "Mensaje enviado con señal de humo: " . $this->senal . PHP_EOL;
        }
    }
}
```

Así que hay que volver a cambiar *MensajeroVeloz* (empiezas a pensar que lo de veloz quizá no sea un nombre muy bueno).

En ese momento *Corazón Bravo* vuelve a gritar. Los cuchillos largos cambian de dirección y ya no se dirigen hacia la tribu. ¡Por fin un pequeño respiro!

El consejo de ancianos y la sabiduría de los Interfaces

Vuelves al campamento y lo consultas con el consejo de ancianos. Ellos te dicen que el problema que estás teniendo se debe a que la clase *MensajeroVeloz* está muy ***acoplada*** a las otras clases. Cuando esto sucede casi cualquier cambio en una de las clases supondrá cambios en *MensajeroVeloz*.

Una clase está acoplada a otra cuando necesita saber mucho de ella. Por ejemplo, en la clase *SenalesHumo* tenemos que dar un valor la propiedad *$mensaje* antes de enviar el mensaje. *MensajeroVeloz* debe "conocer" ya dos cosas de *SenalesHumo*. Debe saber de la existencia del método *enviar()* y de la propiedad *$mensaje*. Cuanto más sepa una clase de otra más acoplada estará.

El gran y sabio *Geronimo* cuenta que en su juventud oyó hablar de los místicos y fabulosos poderes de los *Interfaces*. Nadie los había visto pero su tribu guardaba un pergamino en el que los describían. Como no quieres que vuelva a sucederte lo mismo decides arriesgarte y probar suerte con ellos. Muchos en la tribu tienen miedo... ¿se enfadarán los espíritus si se invoca el poder de los *Interfaces*?

Después de dos días de encerrarte en la tipi sagrada recibiendo visiones sales con el primer *Interfaz* de la tribu:

```
interface Canal
{
    public function enviar($mensaje);
}
```

Desde este momento todos los de la tribu que escriban un sistema para enviar mensajes tendrán que respetar este interfaz.

Después de convocar a *Perro Loco* y ordenarle que cambie sus clases vuelve con ellas modificadas:

```php
class SenalesHumo implements Canal
{
    public function enviar($senal)
    {
        if ($senal) {
            echo "Mensaje enviado con señal de humo: " . $senal . PHP_EOL;
        }
    }
}

class PalomaMensajera implements Canal
{
    public function enviar($mensaje)
    {
        if ($mensaje) {
            echo "Mensaje enviado con paloma mensajera: " . $mensaje . PHP_EOL;
        }
    }
}
```

Y así, ya puedes modificar *MensajeroVeloz* para que funcione sin problemas:

```php
class MensajeroVeloz
{
    private $mensaje;

    public function __construct($mensaje)
    {
        $this->mensaje = $mensaje;
    }

    public function enviarMensaje(Canal $canal)
    {
        $canal->enviar($this->mensajeCifrado());
    }

    private function mensajeCifrado()
    {
        // Trampa, aquí debería ir el código para cifrar el mensaje
        return $this->mensaje;
    }
}
```

A partir de ahora, cuando alguna de las clases *Canal* hagan algún cambio éste no afectará a *MensajeroVeloz* porque éstos deben respetar el interfaz. Ya no importa cuántos métodos extra tengan las otras clases o cómo se llamen sus propiedades, mientras tengan un método *enviar()* todo irá bien.

Pero ¿y qué pasa con la clase *Telegrama*? Esta clase la ha escrito un rostro pálido y la tribu no tiene ningún control sobre ella. Para esta clase decides usar una clase intermedia de la que se encargará *Perro Loco*:

```
class TelegramaQueVuela implements Canal
{
    public function enviar($mensaje)
    {
        $mensajero = new Telegrama;
        $mensajero->enviarTelegrama($mensaje);
    }
}
```

Perrito Chiflado (hijo de *Perro Loco*) quiere iniciarse en las artes chamánicas de la tribu y quiere ver en funcionamiento la clase *MensajeroVeloz*. Así que se lo muestras en una visión:

```
$mensajero = new MensajeroVeloz("¡Alerta! ¡Se acercan enemigos!");

$senalHumo = new SenalesHumo;
$mensajero->enviarMensaje($senalHumo);

$paloma = new PalomaMensajera;
$mensajero->enviarMensaje($paloma);

$telegrama = new TelegramaQueVuela;
$mensajero->enviarMensaje($telegrama);
```

Apunte histórico:

- Cochise no podía usar los sistemas de mensajería porque estaban todos controlados por el enemigo (incluido el ya famoso Google Creek). ¿Que por qué no usó Telegram? ¿Es que no sabes nada de historia? ¡Todavía no había ningún cable intercontinental conectado con el Imperio Ruso y no se podía usar ese servicio!.
- A pesar del tono amable y jocoso de este artículo la historia de los apaches y lo que sufrieron no es cosa de broma. Si tienes interés sobre el tema echa un vistazo a la entrada "Guerras apaches" en la Wikipedia.