Sobre este libro

Objetivos

Este libro se plantea por objetivos enseñarte:

- Las principales características del paradigma de Orientación a Objetos
- Un modo de estructurar tu código de modo de hacerlo fácilmente extensible y adaptable a diferentes situaciones
- Cómo aprovechar al máximo las herramientas específicas de PHP para aumentar tu productividad
- Cómo apalancarte en funcionalidad desarrollada por otros
- Cómo manejar los errores inesperados
- Cómo escribir código reutilizable

Quién debería leerlo

Este libro está escrito pensando en desarrolladores PHP con algo de experiencia y que están buscando elevar su seniority y encarar desafíos de mayor envergadura.

Se asume que conoces las bases de la programación (los conceptos de variables, funciones, estructuras de control, etc...) y estás familiarizado con la sintaxis de PHP.

Cómo está organizado

El libro está organizado en capítulos en los que se aborda un tema específico y se lo estudia en profundidad[1].

Cuenta con ejemplos de código, figuras y llamados a la reflexión para hacer más dinámica su lectura y ayudar a fijar conocimientos.

[1] Salvo el capítulo “Temas avanzados” que hace una breve introducción a una serie de temáticas.

Prólogo

La complejidad de los sistemas informáticos actuales requiere un enfoque diferente del estructurado para hacer frente a los rápidos cambios.

El tamaño de dichos sistemas usualmente requiere la participación de muchas personas en su desarrollo, lo cual vuelve crucial la aplicación de técnicas que permitan la integración del trabajo de un modo claro y sencillo, a fin de dedicar los esfuerzos a superar los problemas de negocio y no a luchar contra las herramientas utilizadas para solucionarlos.

La Programación Orientada a Objetos plantea un modo radicalmente diferente de pensar el software, acercándolo mucho más a la realidad que se intenta modelar y permitiendo una agilidad imposible de lograr mediante el desarrollo estructurado.

Las últimas versiones de PHP abrazan los conceptos fundamentales de esta filosofía, poniendo a disposición del desarrollador un vasto juego de herramientas que hacen su tarea diaria mucho más amena y aumentan significativamente su productividad mejorando a su vez la calidad de sus entregables.

Introducción a la Programación Orientada a Objetos

Qué es la Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es una metodología para escribir los programas de una forma muy cercana a la realidad que se observa.

Es lo que se conoce como un *paradigma* de programación.

Existen diversas formas de entender y resolver un problema pero, independientemente de cuál sea la forma que utilicemos, siempre partiremos de un modelo (Una forma de entender la realidad).

Cada paradigma propone un conjunto diferente de herramientas para construir ese modelo.

El más extendido (antes de la Programación Orientada a Objetos) era el paradigma estructurado.

Sus principales características son:

- Una separación muy fuerte entre datos y procesos
- Una forma de escribir los programas indicando con mucho detalle *cómo* deben ser resueltos los problemas

Existen otros paradigmas (otras formas de escribir programas) basados en reglas muy diferentes (Como la programación funcional o la programación lógica).

POO no es *tan* diferente de la programación "normal" como los otros paradigmas (de hecho es a veces considerada un sub-paradigma), sin embargo, existen conceptos que resultan mucho más sencillos de implementar utilizando POO y otros que directamente no están disponibles en Programación Estructurada.

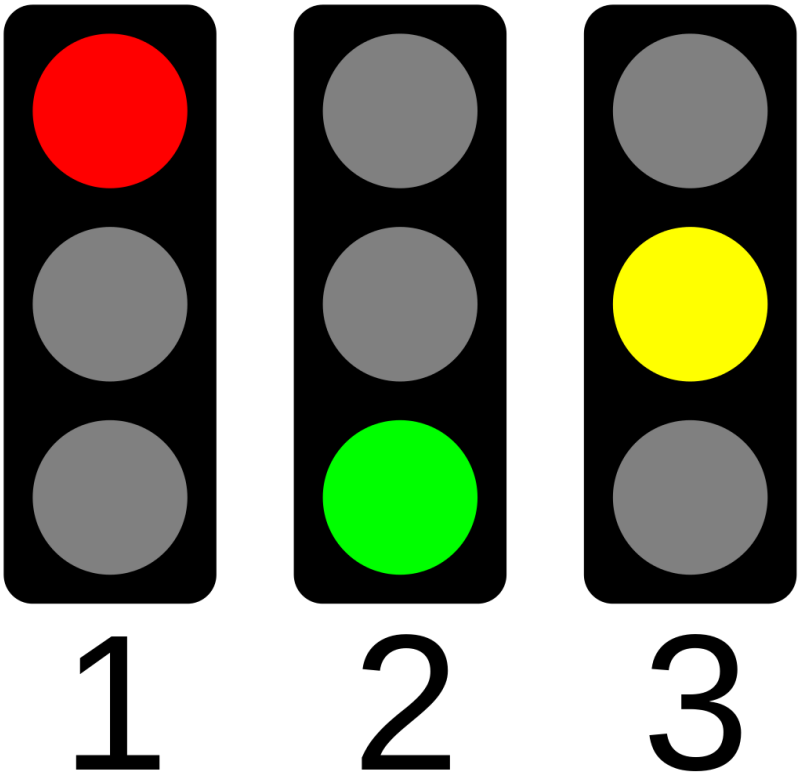
Un mundo de objetos

En POO el concepto principal es el *objeto*.

A diferencia de la programación estructurada, donde las funciones (o procedimientos) eran completamente independientes de los datos, en POO se busca escribir programas que sean mucho más similares a descripciones de universos.

Para ello es importante comenzar identificando cuáles son los actores (o entidades) que interactúan en un contexto determinado y de qué forma lo hacen.

Tomemos por ejemplo un cruce de calles con semáforo.



¿Cómo describirías lo que sucede?

Podrías observar que existen:

- 2 semáforos (Cada uno de los cuales puede tener encendida una luz color rojo, amarillo o verde)
- Autos que se acercan hasta la bocacalle

Si puntualizáramos un poco más veríamos que lo único que hacen los semáforos es cambiar el color de la luz encendida y esperar.

Mientras tanto, los autos pueden acelerar, frenar, abrir sus puertas, etc...

A su vez, las luces de colores *pertenecen* al semáforo, junto con la acción de cambiar la luz activa.

Lo interesante de este sistema es que, para que funcione correctamente, los autos deben avanzar únicamente cuando tienen una luz de color verde a su favor.

Dejando de lado al conductor (O mejor dicho, imaginando que tuviésemos que diseñar un sistema de conducción autónomo), notaríamos que el auto debe *comunicarse* con el semáforo para tomar la decisión de avanzar o quedarse en su lugar.

Un poco de esto se trata la POO. De entender quién se comunica con quién y de qué forma se realiza esa comunicación.

En programación estructurada pensaríamos que hay una función de cambiar la luz activa de un semáforo, en POO diríamos que el semáforo es *reponsable* de cambiar su luz activa.

Reflexiona:

¿Por qué querríamos pensar nuestros programas de esta forma?

Los mensajes

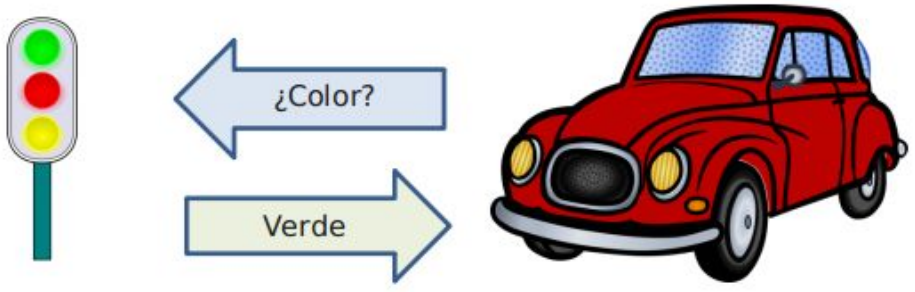
El primero de los conceptos importantes ya lo hemos visto: el objeto.

Luego está el concepto de *mensaje*.

La idea aquí es que, en este mundo de objetos, para lograr un objetivo complejo, estos objetos deben colaborar entre sí y, para ello deben comunicarse.

Esta comunicación se realiza a través del envío y recepción de mensajes (Por ahora trata de no pensar mucho en términos de código... Ya habrá tiempo para eso).

Este intercambio de mensajes crea un *protocolo*.

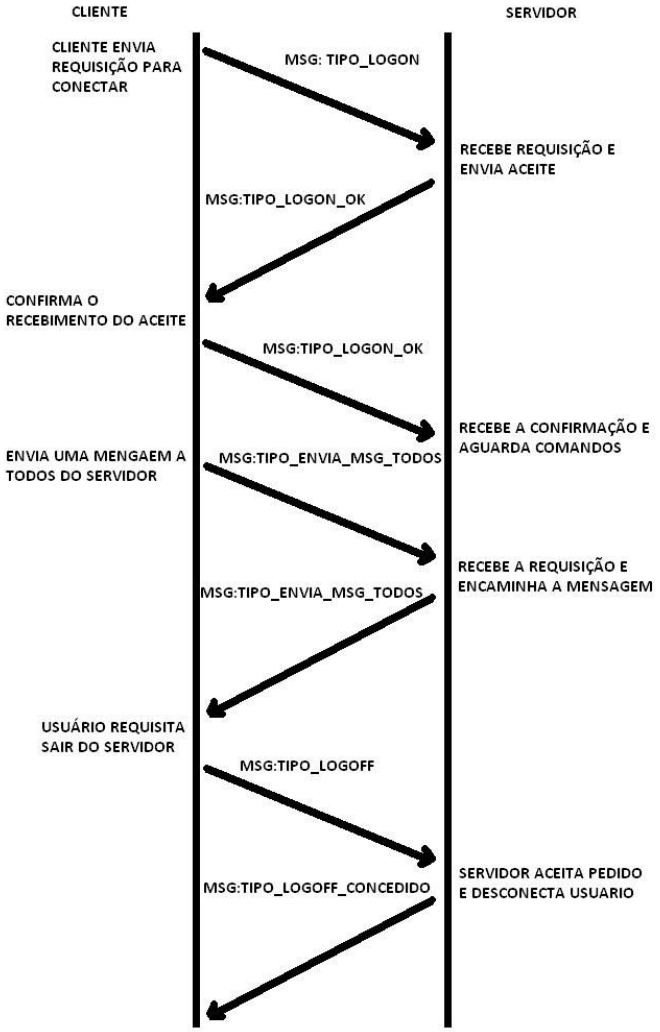


En nuestro ejemplo del semáforo podríamos pensar que el auto pregunta al semáforo de qué color es su luz actual para tomar la decisión de utilizar su capacidad de acelerar o no.

El semáforo contestará “Verde”, en cuyo caso el auto efectivamente avanzará.

Importante
Como en cualquier otro tipo de comunicación, el orden de los mensajes tiene una gran relevancia. No respetarlo puede conducir a que la comunicación no pueda producirse o, peor, a que el resultado sea diferente del esperado

Suele utilizarse esquemas como este para definir los protocolos de comunicación entre objetos (Secuencia de mensajes):



Las clases

Por último, existe otro concepto fundamental de la POO: las clases.

Básicamente, una clase es una descripción *abstracta* de cómo es un conjunto de objetos.

Por ejemplo, si tuvieses que contarle a alguien sobre una silla que viste en un local dudo que tuvieses problemas, ¿cierto?

¿Cómo describirías esta silla?

Seguramente dirías que es de color marrón, de madera, acolchonada con terciopelo... Etc...

Pero, ¿cómo se la describirías a alguien que nunca ha visto una silla en su vida?

Es más complejo, ¿cierto?

De eso se trata el concepto de clase, se trata de pensar **qué características tienen todos los objetos de un determinado tipo** (o conjunto).

En este caso, sería como pensar qué tienen en común todas las sillas que conoces.

Reflexiona:

¿En qué se diferencia describir una silla de explicar qué es una silla?

Esa descripción, en POO, **incluye qué acciones** se pueden realizar con un determinado objeto (¿Para qué sirven esos objetos?).

A su vez, al tener una descripción tan minuciosa, la clase termina siendo una herramienta super importante a la hora de *construir* nuevos objetos del mismo tipo (¿Cómo construirías una silla sin saber cómo se ve una?).

Y eso nos lleva al último de los conceptos: la **instancia**.

Una instancia de una clase es un objeto particular, un ejemplar de un conjunto de objetos similares.

Otro ejemplo:

Se trata de un objeto esférico, hecho de plástico o cuero al que se puede patear o cabecear y es utilizado en diversos deportes.

¿Adivinas de qué clase de objetos estoy hablando?

Veamos algunas instancias:

Apuesto a que ahora ya lo sabes :)

En este caso, el primer objeto está hecho de plástico, el segundo de cuero.

El primero es de color naranja, el segundo blanco.

Pero ambos pertenecen a la misma clase.

En resumen:

Una clase especifica cuáles son las características que todo objeto debe tener para pertenecer a ella.
Una instancia es una asignación particular de valores a esas características.

Las propiedades

Esas características de las que hablamos se conocen en el mundo de la POO con el nombre de **propiedades**.

Pensando un poco más a nivel de implementación, tal vez te hayas dado cuenta de que se tratará en la práctica de variables (Este punto depende mucho del lenguaje particular que vayas a utilizar, pero en el caso de PHP es efectivamente así).

Los métodos

Los métodos son las acciones que los objetos saben hacer. Por ejemplo, un auto sabrá cómo avanzar. Internamente esto implicará hacer girar el motor, lo cual a su vez hará mover las ruedas y eventualmente se logrará lo que se buscaba: avanzar.

Por lo general, la ejecución de un método tendrá algún efecto en las propiedades del objeto.

Volviendo al ejemplo del auto, la acción de avanzar podría afectar, al menos, dos propiedades del auto:

- La posición
- La cantidad de combustible remanente

A su vez, los métodos suelen utilizar los valores de las propiedades para realizar las acciones deseadas.

Por ejemplo, un auto sólo podrá avanzar si cuenta con combustible disponible.

A nivel de implementación, los métodos son funciones que, a diferencia de las funciones de un programa estructurado, pertenecen a una clase (y por lo tanto, a todos los objetos que se creen a partir de ella).

Aquí se va concretizando un poco más el concepto de mensaje: se trata de llamadas a funciones y valores retornados por ellas.

Encapsulamiento

Un concepto sumamente importante que está presente en la POO es el de **encapsulamiento**.

De lo que se trata es de ocultar la mayor cantidad de información posible, de modo que los objetos sean simples de utilizar, aún cuando su funcionamiento interno sea altamente complejo.

Un ejemplo que me gusta utilizar es el de los teléfonos celulares

Hoy en día cualquier persona sabe usar un teléfono celular para una cantidad de cosas diferentes (realizar llamadas, enviar mensajes, leer correos, etc...).

Sin embargo, muy pocos saben cómo toda esa magia efectivamente sucede (Hay una enorme complejidad inherente a la transmisión de datos en forma inalámbrica).

De esta forma, el teléfono se transforma en lo que conocemos como una **caja negra**: sabemos qué debemos suministrarle al aparato y qué obtendremos a cambio,

pero desconocemos (y no nos importa en realidad) qué es lo que el aparato hace para darnos lo que buscamos.

En POO buscaremos que nuestras clases se comporten de este modo: una persona que no participa del desarrollo de la clase debe ser capaz de utilizarla sin necesidad de conocer sus detalles.

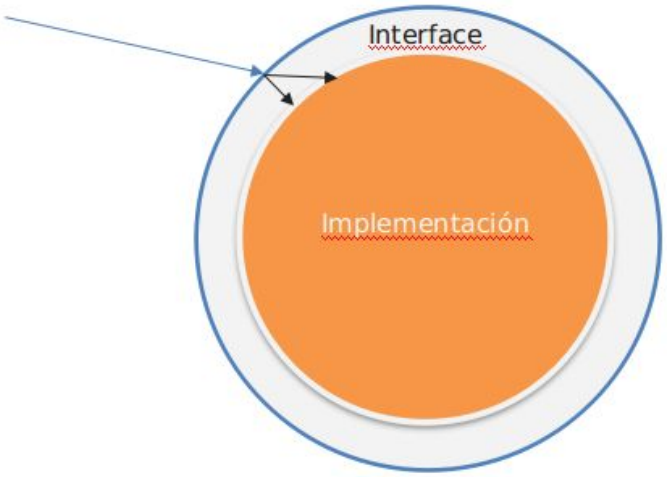
Del principio de encapsulamiento (u ocultamiento de información) se desprenden otros dos conceptos complementarios: Interface e Implementación.

La **interface de una clase es el conjunto de miembros** (propiedades y métodos) que están **disponibles para ser utilizados por código externo** (objetos de otras clases por ejemplo).

La **implementación**, en cambio, es el **conjunto de miembros que sólo están disponibles para ser utilizados por otros miembros de la propia clase**

.

Este pequeño esquema puede ayudar a clarificar las cosas:



Desde el exterior sólo se ve la Interface. Los mensajes que llegan desde afuera van a parar allí y, una vez recibidos e interpretados probablemente generen nuevos mensajes internos (De la interface a la implementación).

Si bien técnicamente interface e implementación podrían ser el mismo conjunto (bastaría con dejar todos los miembros visibles desde el exterior), esta es una muy mala práctica.

Piensa que uno de los objetivos principales de la POO es posibilitar la reutilización de código (Tanto por aplicaciones tuyas como de terceros), lo cual implica que el código será compartido y distribuido.

Esto implica que habrá mucho código que dependa del tuyo, pero, ten en cuenta que ese código se comunicará exclusivamente con la interface de tus objetos.

De algún modo, al crear esta dependencia se está dando un contrato. Tu clase se compromete a devolver siempre un tipo de resultado al recibir un determinado mensaje.

Lo que no está en el contrato sin embargo es cómo se llegará a ese resultado, lo cual da una gran flexibilidad.

Veamos un ejemplo para hacerlo más claro:

Imagina que quieres hacer una aplicación que permita dibujar figuras geométricas.

Muy probablemente quieras tener una clase que modele un punto en el espacio. Típicamente esta representación estará dada por dos valores: X e Y. Imagina entonces que otra persona crea una clase que contiene un método para dibujar líneas utilizando dos objetos de tu clase Punto.

Supongamos entonces que diseñas tu clase de modo de contar con una propiedad X y otra propiedad Y. Pronto te encontrarás con dos opciones:

1. Hacerlas **públicas** (accesibles desde afuera de la clase)
2. Hacerlas **privadas** (accesibles sólo para miembros de la clase) y crear métodos públicos para obtener sus valores.

Probablemente pienses que lo más lógico es tomar la alternativa 1. Analicémoslo un momento.

¿Qué sucederá si, más adelante en el desarrollo, descubres que era más conveniente almacenar la posición en coordenadas polares? (En lugar de X e Y, ángulo y radio).

La función de dibujar una línea dejará inmediatamente de funcionar ya que dependía de la existencia de las propiedades X e Y que no están más...

En cambio, si hubieses ido por la alternativa 2, para que todo siga funcionando simplemente deberías modificar los métodos que permitían obtener los valores de X e Y (Hasta recién eran triviales, ahora requieren algún pequeño cálculo) pero, una vez realizada esa modificación, todo el sistema sigue su funcionamiento normal.

Reflexiona:

Nota como una buena separación entre interface e implementación, aunque un poco más trabajosa al principio, hace la diferencia entre un sistema mantenible y uno no mantenible.

Conclusión

En este módulo has aprendido los conceptos básicos de la Programación Orientada a Objetos (POO).

Has visto cómo este estilo de programación, si bien puede resultarte algo raro al comienzo, es mucho más simple (es mucho más cercano al universo real) y genera código reutilizable casi naturalmente.

Aún falta ver algunas características propias de POO que lo hacen radicalmente diferente de la Programación Estructurada (Conceptos de herencia por ejemplo que estudiaremos más adelante), pero es importante que vayas familiarizándote con la terminología y la forma de pensar de POO.