# Pilares de la POO (parte 1)

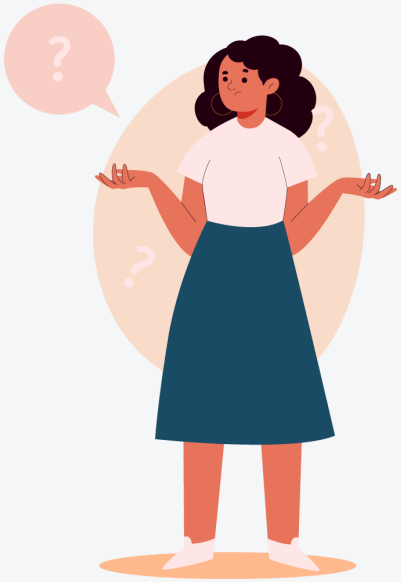

Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1º D
Libro: Pilares de la POO (parte 1)

Imprimido por: Eduardo Moreno
Día: lunes, 7 de abril de 2025, 01:55

# Tabla de contenidos

## Preguntas orientadoras

- ¿Qué es la abstracción en la programación orientada a objetos y por qué es importante?
- ¿Cuáles son los beneficios del encapsulamiento en términos de ocultamiento de información y seguridad del código?
- ¿Cómo se logra la abstracción y el encapsulamiento en la práctica, y qué pautas debes seguir al diseñar tus clases y objetos?
- ¿Cuál es la importancia de la visibilidad en la encapsulación y el diseño modular?



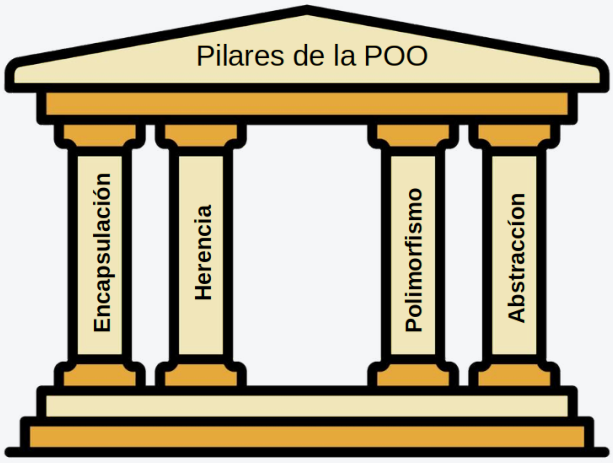
## Pilares de la POO

Tal como hemos trabajado en la unidad anterior, la POO se basa en objetos de la realidad para modelar o diseñar clases que se relacionan entre sí. Estas clases sirven de plantilla o modelo para la creación de objetos que forman parte de la aplicación informática.

Pero, ¿por qué retomamos esta idea? La POO se ha convertido en una técnica muy popular de programación, debido a sus ventajas respecto a métodos tradicionales. Estas ventajas se deben a la idea de separar las parte públicas y privada de una solución y a sus cuatro pilares fundamentales, ¿te acordás que vimos los dos primeros? Pero, ¡hay más!

### Los pilares de POO

- Abstracción
- Encapsulación
- Herencia
- Polimorfismo




**A continuación, retomaremos los pilares abstracción y encapsulamiento.**

## Abstracción

### ¿Qué significa "abstraernos" en POO?

La abstracción en la programación orientada a objetos (POO) es un concepto clave que permite representar entidades del mundo real como objetos en el código. En otras palabras, proporciona una forma de modelar y organizar la información relevante y las funcionalidades asociadas a un objeto, ocultando los detalles internos y centrándose en los aspectos esenciales.

La abstracción es la propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista determinado y no focalizar en los restantes aspectos, es decir, se refiere al hecho de diferenciar entre las propiedades externas de un objeto y los detalles de la composición interna de dicho objeto.

En conclusión, la abstracción permite ignorar los detalles internos de elementos complejos como una computadora, un auto, un televisor, etc.

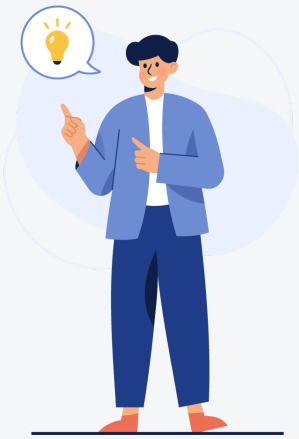
### Niveles de abstracción

La abstracción posee diversos niveles que ayudan a estructurar la complejidad de los sistemas del mundo real. En el análisis de un sistema es importante concentrarse en ¿Qué hace? y no en ¿Cómo lo hace?

A continuación, relacionemos esta idea con un ejemplo.

**¡Veamos un ejemplo!**

Para entender mejor el concepto de la abstracción es útil hacer analogía con el mundo real. Por ejemplo, un televisor suele manejarse con un control remoto que tiene las funciones de encender, apagar, cambiar de canal, ajustar el volumen y también nos permite añadir componentes externos como parlantes, reproductores de DVDs, conexión a Internet, etc. Sin embargo, no necesitamos saber cómo funciona internamente, cómo recibe la señal, ni cómo la muestra en pantalla. Simplemente sabemos cómo utilizarlo.



Esta característica se debe a que el televisor separa claramente su implementación interna de su interfaz externa.

Durante el proceso de abstracción es cuando se decide qué características y comportamiento debe tener el modelo.

**¡Volvemos sobre la idea de niveles!** Aplicando la abstracción podemos construir, analizar y gestionar sistemas grandes y complejos que no se podrían diseñar si se tratara de modelar a un nivel detallado.

En cada nivel de abstracción se visualiza el sistema en términos de componentes abstractos, cuya composición interna se ignora. Esto nos permite concentrarnos en cómo cada componente interactúa con otros componentes y centrarnos en la parte del sistema que es más relevante para la tarea a realizar en lugar de perdernos a nivel de detalles menos significativos.

## ¿Cómo se aplica la abstracción?

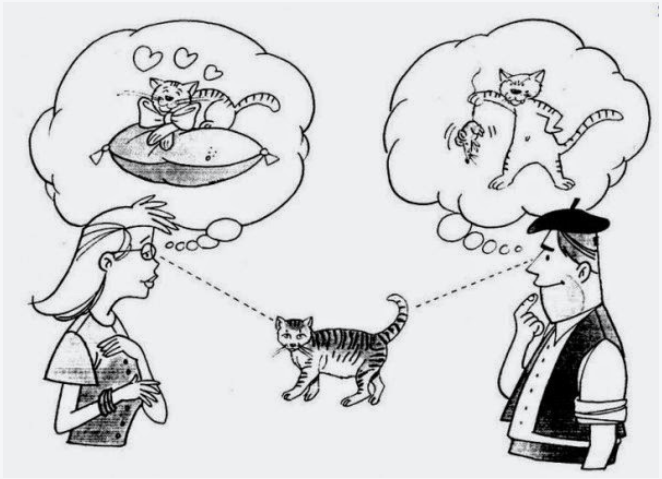
La abstracción se logra a través de la creación de clases y la definición de sus atributos y métodos.

¡Repasemos! Como vimos, una clase es una plantilla o molde que describe las características y el comportamiento de un objeto. Por ejemplo, si estamos creando un sistema para una biblioteca, podemos tener una clase llamada Libro que representa un libro en sí mismo.

La abstracción nos permite enfocarnos en los aspectos importantes de un objeto y omitir los detalles irrelevantes. Por ejemplo, en la clase Libro, podemos definir atributos como "título", "autor" y "año de publicación". Estos atributos encapsulan la información necesaria para describir un libro, mientras que los detalles internos, como el método de almacenamiento de la información o la implementación específica de los atributos, se ocultan.

Además de los atributos, la abstracción también involucra la definición de métodos. Los métodos representan las acciones o comportamientos que un objeto puede realizar. Siguiendo el ejemplo de la clase Libro, podríamos tener métodos como "prestarLibro()" o "devolverLibro()", que definen cómo interactuar con el libro en el contexto de una biblioteca pero seguramente tendrá otros métodos si se tratase de un estudiante que interactúa con un libro o incluso otros si lo que estamos modelando es una librería on-line.

En conclusión, la abstracción en POO nos permite crear objetos que son representaciones simplificadas de entidades del mundo real. A través de la abstracción, podemos modelar objetos con sus propiedades y comportamientos esenciales, y utilizarlos en nuestro programa sin tener que preocuparnos por los detalles internos de su implementación. Esto nos permite desarrollar software más modular, fácil de mantener y reutilizable.



Continuemos con el siguiente pilar: encapsulamiento.

## Encapsulamiento

El encapsulado de datos es el proceso de agrupar datos y operaciones relacionadas bajo la misma unidad de programación. En el caso de los objetos que poseen las mismas características y comportamiento se agrupan en clases, que no son más que unidades o módulos de programación que encapsulan datos y operaciones.

### Acerca del acceso y protección de datos

La ocultación de datos permite separar el aspecto de un componente, definido por su interfaz con el exterior, de sus detalles internos de implementación. El acceso al objeto está restringido sólo a través de una interfaz bien definida.

La propiedad de ocultar los datos abstraídos y aislarlos o protegerlos de quien no se desee que tenga acceso a ellos; asegura que los aspectos externos de un objeto se diferencien de sus detalles internos.

¡Tomemos nota! ¿Por qué es importante este pilar? El encapsulamiento es la característica de un lenguaje POO que permite que todo lo referente a un objeto quede aislado dentro de este. Es decir, que todos los datos referentes a un objeto queden "encerrados" dentro de este y solo se puede acceder a ellos a través de los miembros que la clase proporciona (propiedades y métodos).

En C# (y en los demás lenguajes de programación) esto se sostiene a través de la propiedad de visibilidad de cada método y atributo.

A continuación, profundicemos sobre este último punto: la **visibilidad**.

## Visibilidad

### Lo público y lo privado

Cuando se define y modela una clase, una de las actividades más importantes es diferenciar la parte/interfaz pública y la parte privada. La parte pública es aquella a la que todo el código, otras clases, otros algoritmos pueden tener acceso. Es aquella sección que se comunica con el resto del mundo y está abierta a su uso. La parte privada es aquella que se usa internamente, es super necesaria pero no se requiere que sea conocida ni que todo el resto del programa tenga acceso.

Por ejemplo, no es necesario conocer cómo realmente funciona una computadora por dentro para usarla, solo hay que saber cómo funciona su interfaz, su parte publica.

| Parte pública | Parte privada |
|---|---|
| pantalla | drivers y gpu |
| teclado | caché del procesador |
| mouse | interfaz sata discos rigido |
| parlante | interfaz m2 discos rigido |
| fuente de alimentación | dispositivo controlador de batería de notebook |
| apagar() | electrónica manejo de señales wifi |
| prender() | microprocesadores |
| | Northbridge |
| | Southbridge |
| | Memoria |
| | controladores de memoria |
| ..... | ..... |

Obviamente si se abre (físicamente) la computadora se tendrá acceso a todo. Sucede lo mismo si se abre el código fuente del archivo de una clase, se puede modificar todo: pero la razón principal por la cual se hace todo lo anterior es para organizar mejor el código, todo es para mejorar la operación general de una clase ocultando la información que no es necesario que conozcamos y exponiendo la que sí.

## Niveles de visibilidad

En el lenguaje de programación C#, la visibilidad se refiere a la accesibilidad de un miembro de una clase o estructura desde otros elementos del programa. Determina qué partes del código pueden acceder y utilizar dicho miembro.

En C#, hay varios niveles de visibilidad que se pueden aplicar a los miembros de una clase. Estos niveles de visibilidad son especificados mediante modificadores de acceso.

Los modificadores de acceso en C# son los siguientes:

| Modificadores de acceso en C# |
| --- |
| Public: Es el nivel de visibilidad más amplio. Un miembro declarado como público es accesible desde cualquier parte del programa, tanto desde dentro de la clase que lo define como desde otras clases que la utilicen.<br><br>Por ejemplo, si tienes una clase llamada "Persona" con un método público llamado "ObtenerNombre", puedes llamar a ese método desde cualquier otra clase en tu programa. |
| Private: Es el nivel de visibilidad más restrictivo. Un miembro declarado como privado solo es accesible desde dentro de la clase que lo define. No puede ser accedido desde ninguna otra clase.<br><br>Los miembros privados se utilizan para encapsular y ocultar detalles internos de una clase, evitando su acceso directo desde el exterior. |
| Protected: Un miembro declarado como protegido es accesible desde dentro de la clase que lo define y también desde las clases derivadas de esa clase. Esto significa que los miembros protegidos pueden ser utilizados por las clases hijas, pero no por otras clases fuera de la jerarquía de herencia.<br><br>Este tema lo veremos más adelante |
| Internal: Un miembro declarado como interno es accesible desde cualquier parte del mismo ensamblado (assembly). Un ensamblado en C# puede ser un proyecto, una biblioteca o un componente compilado.<br><br>Esto significa que los miembros internos solo pueden ser accedidos desde dentro del mismo proyecto o biblioteca, pero no desde proyectos o bibliotecas externas. El acceso interno es útil para limitar la visibilidad de los miembros a un conjunto específico de componentes dentro de una solución. |
| Protected Internal: Este nivel de visibilidad combina las características de los modificadores de acceso protegido e interno. Un miembro protegido interno es accesible desde cualquier parte del mismo ensamblado y también desde las clases derivadas, ya sea dentro o fuera del ensamblado. |

**¡Toma nota!** Resulta importante tener en cuenta que estos modificadores de acceso se aplican a los miembros de una clase, como variables, métodos, propiedades y eventos. Al utilizar adecuadamente los

niveles de visibilidad, puedes controlar el acceso a los miembros de tus clases y estructurar tu código de manera segura y eficiente.

Al aplicar los diferentes modificadores de acceso a los miembros de una clase, puedes controlar qué partes del código tienen acceso a ellos y qué partes están restringidas. Esto ayuda a garantizar la encapsulación y la seguridad de tu código, al permitir un acceso controlado a los datos y funcionalidades relevantes.

### En resumen

La abstracción y el encapsulamiento son dos conceptos fundamentales en la programación orientada a objetos (POO).

La abstracción es el proceso de simplificar y representar un objeto o sistema de forma conceptual, centrándose en los aspectos relevantes y ocultando los detalles innecesarios. En la POO, se logra mediante la creación de clases, que son plantillas para la creación de objetos. La abstracción permite al programador enfocarse en las características esenciales y comportamiento de un objeto, sin preocuparse por su implementación interna.

El encapsulamiento, por otro lado, se refiere a la ocultación de los detalles internos de un objeto y la exposición controlada de sus funcionalidades a través de métodos y propiedades. En la POO, se utilizan los modificadores de acceso (como público y privado entre otros) para definir qué partes de un objeto pueden ser accesibles desde el exterior. El encapsulamiento permite mantener la integridad y consistencia de los datos al restringir el acceso directo a ellos y proporcionar interfaces claras y seguras para interactuar con el objeto.

**La abstracción simplifica y representa los objetos y sistemas de manera conceptual, mientras que el encapsulamiento oculta los detalles internos y expone solo las funcionalidades necesarias. Ambos conceptos son fundamentales para lograr un diseño de software modular, mantenible y fácil de entender.**