Programación Orientada a Objetos (POO)

Introducción

La *programación orientada a objetos* es el paradigma en el cual los elementos de primer orden son los objetos.

Como *paradigma* es un derivado de la programación estructurada, por el cual se logró que los **datos** y los **métodos** que manipulan esos datos, se mantengan juntos en una unidad llamada **objeto**.

De esta manera se propuso impedir el manejo de datos *salvo mediante llamadas a los métodos del objeto contenedor de los mismos*.

Puede cambiarse la estructura interna de un objeto sin afectar a aquellos que interactúen con él, siempre y cuando no se afecte su interfaz pública.

Objetos

El término 'objeto', como dice Meyer en *Touch of Class*, es uno muy vago y vulgar. En nuestro idioma sólo hay una forma más vaga y vulgar de referirse a los objetos, y es mediante la palabra **'cosa'.**

Pueden amoldarse al concepto que se desee representar, sea éste de cualquier índole.

No necesita existir en el mundo real, ser tangible, ni ser pequeño, o demasiado grande.

Estamos ante la posibilidad de crear y dar forma a **cualquier concepto que imaginemo**s, traduciéndolo a la apariencia de 'objetos' para poder utilizarlos como ***pequeñas máquinas de software que fundamentalmente saben cosas, y saben hacer cosas*.**

Solamente debe importarnos **qué cosas pueden hacer los objetos por nosotros**, y dejamos de lado (por el momento o definitivamente, según el caso) el conocimiento de las cosas que 'saben' los objetos.

Miembros

Lo que hace a un objeto, en palabras de Meyer en *Touch of Class*, no es su contrapartida en el mundo físico, sino que **podemos manipularlo a través de un grupo de operaciones bien definidas a las cuales llamamos miembros.**

Los miembros de un objeto se invocan por medio de mensajes, y suele ser común confundir mensajes con miembros. Haremos una tosca pero práctica distinción:

- **Miembro** es cada una de las posibilidades de manipulación que nos proporciona un objeto. Se evidencia en tiempo de codificación.
- **Mensaje** es la invocación efectiva de esos miembros. Se evidencia en tiempo de ejecución.

Mensajes

En el año 1967 Ole-Johan Dahl y Kristen Nygaard lanzaron el lenguaje de programación Simula 67, el ancestro común de todos los lenguajes de programación orientada a objetos.

Una de las principales innovaciones que introdujeron a la anterior programación estructurada fue convertir esto

f(o,x)

en esto

o.f(x)

La semántica introducida es el componente que cambia las reglas del juego.

> En el caso de f(o,x) estamos diciendo "realizar la función *f* con *o* y *x*".
>
> En cambio con o.f(x) estamos diciendo "objeto *o*, hacé *f* con *x*".
>
> Y aquí hemos cambiado un paradigma.

Mensajes

A partir de la programación orientada a objetos pasamos a depender de mensajes, que estratégicamente implementados nos derivan a los objetos adecuados sin necesidad de ejercer control directo.

Cuando pasamos un mensaje, perdemos el control de quién lo va a interpretar.

Sólo podemos esperar que el receptor reaccione apropiadamente.

Ni el que envía depende del que recibe, ni viceversa.

La acción en la programación orientada a objetos se inicia por la transmisión de un mensaje a un agente (un objeto) responsable por las acciones. El mensaje codifica el pedido de una acción y se acompaña por cualquier información adicional (argumentos/parámetros) necesaria para llevar adelante el pedido. El receptor es el objeto al cual el mensaje está dirigido. Si éste acepta el mensaje, está aceptando la **responsabilidad** de llevar adelante la acción indicada. En respuesta a un mensaje, el receptor llevará a cabo un **método** para satisfacer el pedido.

Mensajes: Consultas y Comandos

Consultas: Los miembros que nos permiten obtener propiedades de un objeto se denominan consultas: estamos *consultando* al objeto por una propiedad en particular.
Según el **principio de acceso uniforme**, todos los servicios ofrecidos por un módulo deben estar disponibles por medio de una notación uniforme, que **no develе** si son implementadas mediante almacenamiento o cálculo de lo requerido.
Tomaremos por ejemplo el siguiente caso:

```
class Persona {
  Integer calcularEdad() { ... }
  Date getFechaNacimiento() { ... }
}
```

Nos damos cuenta que traicionamos el ocultamiento de información: se sabe a ciencia cierta que lo que se almacena es la fecha de nacimiento, y que la edad se calcula.
Respetando el principio de acceso uniforme, deberíamos hacerlo de la siguiente manera:

```
class Persona {
  Integer getEdad() { ... }
  Date getFechaNacimiento() { ... }
}
```

Para devolver los valores las consultas pueden requerir de ciertos parámetros adicionales. Lo importante es que las consultas **no alteran el estado de los objetos**, por lo que eso las define.

Mensajes: Consultas y Comandos

Comandos: Los comandos son peticiones de acción hacia un objeto. Son la contrapartida de las consultas ya que se caracterizan por **modificar el estado de los objetos**.

Un ejemplo podría ser el siguiente:

```
class Auto {
  void apagarLuces() { ... }
  void encenderLuces() { ... }
}
```

Como podemos ver, es evidente que algo cambiará en el estado interno del Auto para que ésto pueda llevarse a cabo. En un análisis más sutil, vemos que el cambio genera que la consecutiva invocación de uno de los miembros deje de tener sentido: *no sirve apagar una luz apagada*.

Ese cambio de estado distingue a los comandos, y nos alerta de que en diferentes instantes de tiempo las invocaciones a consultas o comandos puedan brindar resultados diferentes.
En una frase, el resultado de consultas y comandos siempre depende del estado del objeto, aunque no siempre será modificado.

Clases

Cuando nos referimos a objetos estamos contando a los individuos de un universo repleto de variedad: al hablar de una pelota, no hablamos del concepto general de pelota, sino de una pelota en particular. Será mi pelota, o puede ser otra: cada una con características particulares.

Ese conjunto de características son las que definen el tipo de entidad del que estamos hablando: distinguimos los individuos entre sí por la variabilidad de sus características, pero al referirnos al concepto genérico y colectivo estamos hablando de características sin pensar en ninguna en particular: toda persona tiene altura, aunque no sabemos cuánta hasta pensar en una en particular.

Esta distinción entre individuo y colectivo es la que debemos hacer al pensar en objetos y clases: los objetos son los individuos, y mediante una abstracción de características comunes obtenemos las clases.

Las clases nos permiten tener un molde para crear objetos a partir de ellas. Imaginemos que una clase es una plantilla que posee algunos huecos para completar a gusto: los completaremos cuando apliquemos la plantilla para crear un nuevo objeto.

Las clases tienen tres características:

- nombre
- atributos
- operaciones.

Clases

Cada individuo (que posee identidad) de determinada clase dará valores a esos atributos y comportamiento efectivo a esas operaciones.

Al definir una clase damos la posibilidad de la existencia de los objetos:

```
public class Estudiante {
  // atributos, constructor, métodos
}
```

Nos permite escribir:

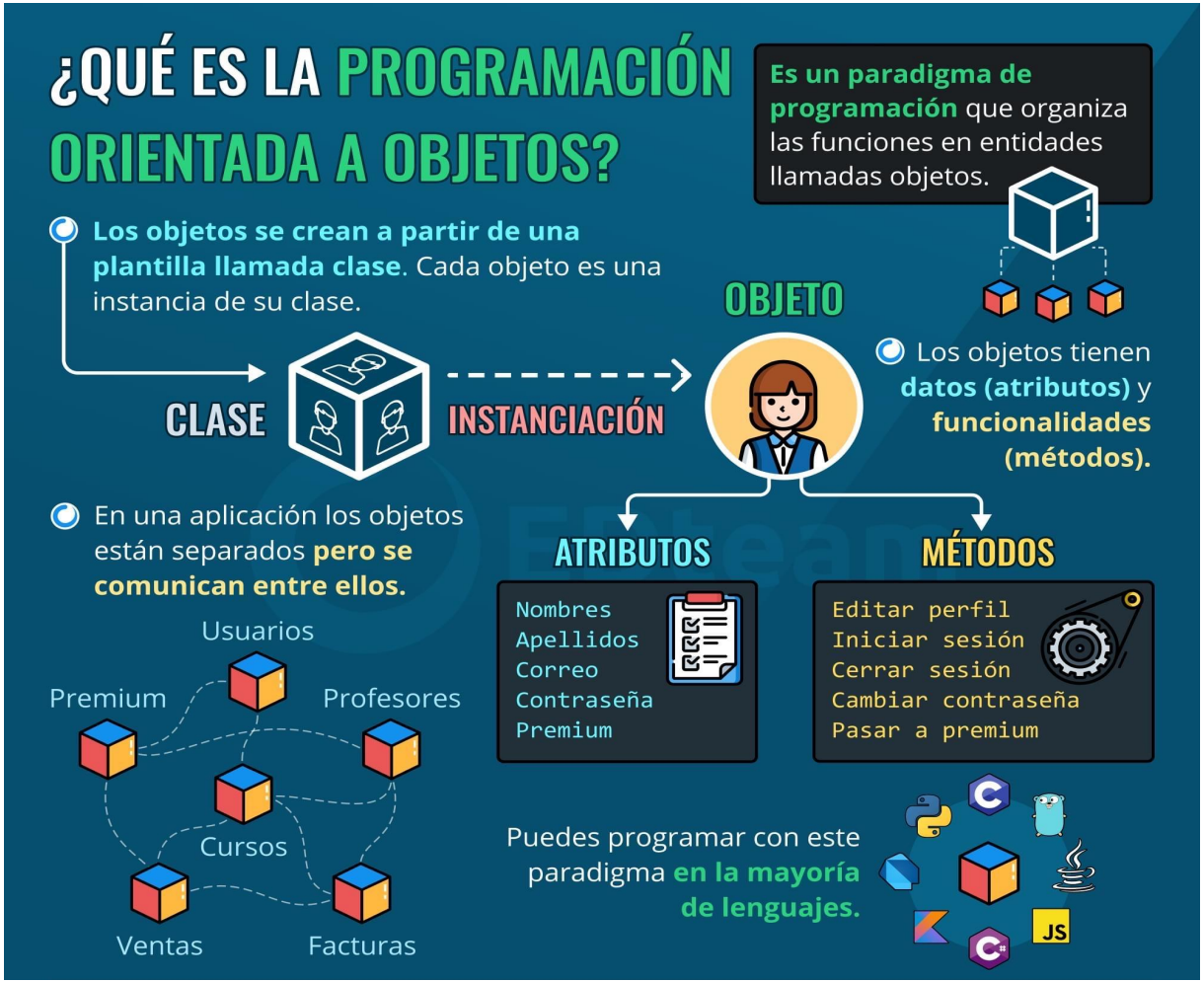
```
Estudiante unEstudiante = new Estudiante("Juan");
```

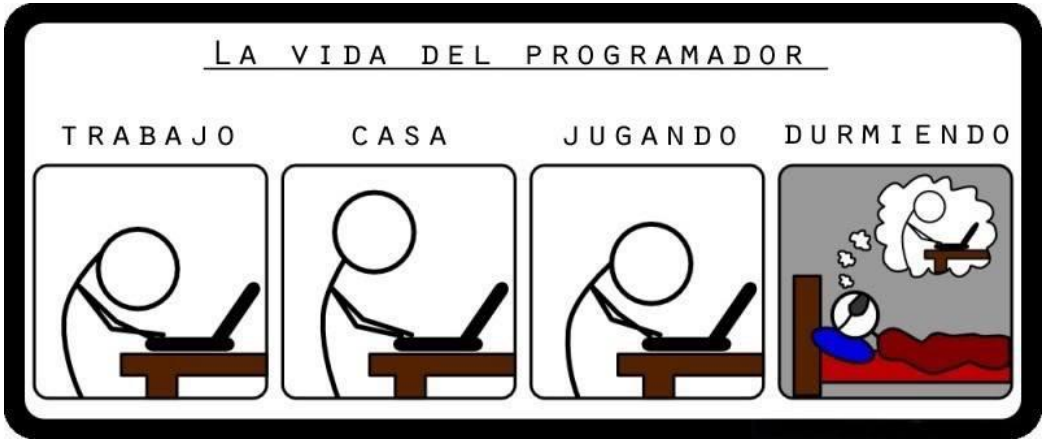
En esa simple sentencia tenemos la relación que existe entre un objeto y una clase. "Un estudiante es una nueva instancia de Estudiante, en este caso su nombre será Juan".

Los objetos tienen identidad y un estado interno particular. La clase no tiene identidad, ya que es un molde para todos los objetos, ni tiene valores sino "espacios" preparados para adquirirlos.

Si bien los objetos son los que reciben los mensajes y ejecutan acciones, las clases imponen qué instrucciones deberán ejecutarse y de qué modo en cada caso.

Resumen


¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?
Es un paradigma de programación que organiza las funciones en entidades llamadas objetos.
Los objetos se crean a partir de una plantilla llamada clase. Cada objeto es una instancia de su clase.
CLASE
INSTANCIACIÓN
OBJETO
Los objetos tienen datos (atributos) y funcionalidades (métodos).
En una aplicación los objetos están separados pero se comunican entre ellos.
Usuarios
Premium
Profesores
Cursos
Ventas
Facturas
ATRIBUTOS
Nombres
Apellidos
Correo
Contraseña
Premium
MÉTODOS
Editar perfil
Iniciar sesión
Cerrar sesión
Cambiar contraseña
Pasar a premium
Puedes programar con este paradigma en la mayoría de lenguajes.
JS


La vida del programador
TRABAJO
CASA
JUGANDO
DURMIENDO