1 ¿Qué es la Programación Orientada a Objetos (POO)?

Mucho se ha escrito desde que surgiera esta nueva forma de programar. Muchos autores han definido con mayor o menor exactitud que es la **POO.**

POO: **Según dice el Dr. Timothy Budd (profesor asociado de Ciencias de la Computación en Oregon State University), en su libro "Introducción a la Programación Orientada a Objetos", la POO *es una nueva forma de pensar acerca de lo que significa computar, acerca de cómo podemos estructurar la información dentro de un computador.***

La POO no es:

- **Una forma de comunicarse con los programas basados en ratones, ventanas, iconos, etc., es decir, entornos gráficos.** Aunque los lenguajes de POO suelen presentar estas características y estos entornos suelen desarrollarse con técnicas de **POO**, algunas personas tienden a identificar POO con entornos de este tipo.

- **No es un lenguaje.** Las técnicas de **POO** se pueden usar con cualquier lenguaje existente, aunque en los lenguajes que usan estas técnicas, como por ejemplo Java, la implementación es mucho más fácil y provechosa.

2 Un paradigma de la programación

En ambientes académicos se alude a la **POO** como un **nuevo paradigma de programación.** Con anterioridad a éste, son también paradigmas de programación, la Programación Imperativa (Pascal o C), la Programación Lógica (Prolog), y la Programación Funcional (ML).

La palabra paradigma (del Latín), significaba originalmente, ejemplo ilustrativo, enunciado modelo que mostraba todas las inflexiones de una palabra, pero el historiador Thomas Kuhn extendió la definición de la palabra para abarcar un **conjunto de teorías, estándares y métodos que juntos representan una forma de organizar el conocimiento,** esto es, una forma de ver el mundo. Es en éste último sentido que la **POO** es un nuevo paradigma.

La **POO** nos obliga a reconsiderar nuestras ideas acerca de la computación, de lo que significa ponerla en práctica y de cómo debería estructurarse la información dentro del computador.

"La POO son un conjunto de técnicas que permiten incrementar nuestro proceso de producción aumentando la productividad por un lado y permitiendo abordar proyectos de mucha mayor envergadura por otro".

Usando estas técnicas, nos aseguramos la **re-usabilidad** de nuestro código, es decir, lo que hoy escribimos, si está bien escrito, nos servirá para *"siempre".*

3 Una manera de ver el mundo

Para ilustrar algunas de las principales ideas de la **POO** vamos a considerar una **situación de la vida real** para preguntarnos después como lograríamos que el ordenador modelara las técnicas empleadas.

ejemplo	Supongamos que deseo enviar flores a mi abuela (que se llama María) con motivo de su cumpleaños. María vive en una lejana ciudad a muchos kilómetros, por lo que es imposible que yo misma recoja y lleve las flores hasta su puerta. Sin embargo, enviarle flores es una tarea fácil: sólo voy a la florista local (que se llama Rosa), describo la **clase** y cantidad de flores que quiero enviar, y puedo estar seguro de que las flores serán entregadas sin más. El mecanismo que usé para resolver mi problema fué encontrar un **agente apropiado** (es decir, Rosa), y pasarle un **mensaje** que contuviera mi **petición.** Es responsabilidad de Rosa satisfacer **mi solicitud.** Para ello Rosa empleó un **método,** es decir, un **algoritmo o conjunto de operaciones.** No necesito saber qué método particular usará Rosa para satisfacer mi petición. No nos interesan los detalles. Sin embargo, si decidiera investigar como la hará, tal vez descubriría que Rosa entrega un **mensaje** ligeramente diferente a otra florista, de la ciudad donde vive mi abuela. Dicha florista, a su vez, hace el arreglo y lo pasa, junto con otro **mensaje,** a un repartidor, y así sucesivamente. De esta manera, el vehículo por el cual son iniciadas las actividades constituye nuestro primer principio de la resolución de problemas orientados a objetos:

En la POO, la acción **se inicia mediante la transmisión de un mensaje a un agente** (un objeto) responsable de la acción.
El **mensaje** tiene codificada la petición de una acción y se acompaña de cualquier información adicionjal (argumentos) necesaria para llevar a cabo la petición.
El **receptor** es el agente al cual se envía el mensaje. Si el receptor acepta el mensaje, acepta la responsabilidad de llevar a cabo la acción indicada. **En respuesta a un mensaje, el receptor ejecutará algún método para satisfacer la petición.**

mensajes y métodos

Destaca, en el paso de mensajes, el **pricipio de ocultación de la información.** Esto es, el cliente que envía la petición no necesita conocer el medio real con el que ésta será atendida.

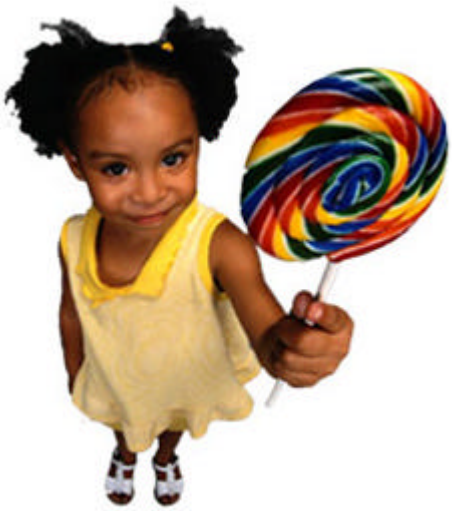
Por otro lado, ¿en que **difiere** un mensaje **de una llamada a un procedimiento**? En ambos casos hay una petición implícita de acción. En ambos, hay un conjunto de pasos bien definidos que se iniciarán después de la petición. Pero hay dos diferencias importantes:

- La primera es que el mensaje posee un receptor designado, el receptor es algún agente al cual se envía el mensaje (en nuestro caso Rosa). La llamada a un procedimiento, en cambio, carece de un **receptor designado.**

- La segunda es que la **interpretación del mensaje** (es decir, el método usado para responder al mensaje) depende del receptor y puede variar con diferentes receptores.

Por ejemplo, si doy el mismo mensaje a mi esposa Inma, ella lo entenderá y se obtendrá un resultado satisfactorio. Sin embargo, el método que Inma use para satisfacer la petición, será diferente del ejecutado por Rosa. Si le pidiera a Carlos, mi dentista, que enviara flores a mi abuela, quizás estaría cometiendo un error, porque puede ser que Carlos no tenga un método para resolver el problema.

Es muy importante notar que el comportamiento se estructura en términos de las responsabilidades. Mi solicitud de acción sólo indica el resultado deseado (flores para mi abuela). La florista es libre de seguir cualquier técnica que logre el **objetivo deseado.**



Aunque hayamos tenido poco contacto con Rosa, tenemos una idea aproximada del comportamiento que podemos esperar al entrar en su negocio y exponerle la petición. Puedo hacer ciertas suposiciones porque sé algo de los floristas en general y espero que Rosa, por ser un ejemplar de esta categoría, entrará en el patrón general.

Podemos usar el **término Florista para representar la categoría** (o **clase**) **de todos los floristas.**

Todos los **objetos son ejemplares de una clase.** El método invocado por un objeto en respuesta a un mensaje queda determinado por la clase del receptor. Todos los objetos de una clase dada usan el mismo método en respuesta a mensajes similares.	clases y ejemplares

Tenemos además más información acerca de Rosa, no necesariamente porque sea florista, sino porque es comerciante.

Sé, por ejemplo, que probablemente me pedirá dinero como parte de la transacción y a cambio del pago me dará un recibo.

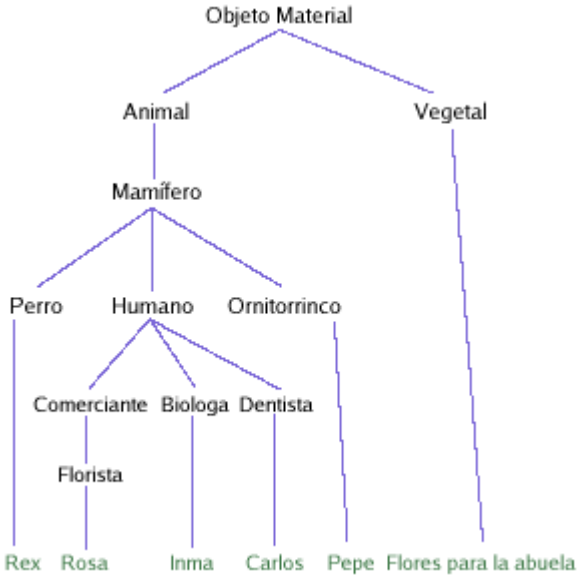
Tales acciones son válidas para tenderos, libreros y otros comerciantes. Puesto que la categoría **Florista es una forma más especializada de la categoría Comerciante,** cualquier conocimiento que yo tenga de los Comerciantes también es válido para los Floristas y por lo tanto, para Rosa.
Una forma de pensar en cómo he organizado mi conocimiento acerca de Rosa es en términos de una jerarquía de categorías.

Rosa es una Florista, pero Florista es una forma especializada de Comerciante. Más aún, un Comerciante es también un Humano; así yo sé, por ejemplo, que Rosa es probablemente bípeda. Un Humano es un Mamífero (por lo tanto, amamanta a sus crías), un Mamífero es un Animal (por lo tanto respira oxígeno), y un Animal es un Objeto Material (por lo tanto, tiene masa).

De esta manera, gran parte del conocimiento que tengo que es aplicable a Rosa no está directamente asociado con ella, o siquiera con su categoría de Florista. El principio que dice que el **conocimiento de una categoría más general es aplicable también a la categoría más específica se llama herencia.** Decimos que la clase Florista **heredará atributos de la categoría Comerciante.**

herencia

Las **clases** se pueden organizar en una estructura de herencia jerárquica. Una subclase heredará atributos de una superclase que esté más arriba en el árbol. Una superclase abstracta es una clase (como Mamífero) que se usa sólo para crear subclases y para la cual no hay ejemplares directos.



Pepe el ornitorrinco representa un problema para nuestra estructura de organización simple. Por ejemplo, sabemos que los mamíferos dan a luz, y Pepe ciertamente es un mamífero, y aún así Pepe (o más bien su pareja, Pepita) sigue poniendo huevos. Así es que necesitamos encontrar un **método para resolver estas excepciones.**

Lo hacemos decretando que la información contenida en una subclase puede anular información de una superclase.

Normalmente para solucionar este problema haremos que un **método de una subclase tome el mismo nombre que el método de la superclase,** este método contendrá

la información suficiente para resolver el problema del ornitorrinco.

La búsqueda para encontrar un método que pueda invocarse en respuesta a un mensaje dado empieza con la clase del receptor. Si no se encuentra un método apropiado, se lleva la búsqueda a la superclase de dicha clase. La búsqueda continúa hacia arriba de la cadena de la superclase hasta que se encuentra un método o se agota la cadena de la superclase. En el primer caso, el método se ejecuta; en el último caso, se emite un mensaje de error.	enlace de métodos

Es decir, si Rosa no encontrara el método apropiado dentro de la clase Florista, entonces se buscaría dentro de la clase Comerciante y así sucesivamente dentro del árbol jerárquico hasta encontrarlo.

El hecho de que mi esposa Inma y la florista Rosa, respondan a mi mensaje ejecutando diferentes métodos es un ejemplo de polimorfismo.

4 Manejo de la complejidad y mecanismo de abstracción

Hace unas décadas la programación se realizaba en **lenguaje Ensamblador.** Los programadores trabajaban solos en el desarrollo de programas que hoy en día consideraríamos pequeños.

Estos programas tenían muchas líneas de código y obligaban a sus desarrolladores a hacer un gran esfuerzo para recordar el uso de variables, de posiciones de memoria, etc. Surgían problemas como ¿Qué valores contenían los registros del procesador? ¿Que variables tengo que inicializar antes de ceder el control a aquel otro bloque de código?...

La aparición de los **lenguajes de Alto Nivel** evitó el uso de mnemotécnicos a los programadores y les permitió abstraerse de la complejidad inicial, cercana al procesador, de aquellos lenguajes ensambladores.

Los lenguajes de Alto Nivel estaban dotados de instrucciones sencillas de manejar, con un formato **más cercano a un lenguaje natural.**

Dos de las dificultades más importantes que se salvaron fueron el manejo automático de variables locales, y la concordancia implícita de argumentos de parámetros. Incrementaron las expectativas de la gente acerca de lo que podría hacer un computador, de tal forma que **provocó el surgimiento de un mayor número de problemas.**

En la medida en que los programadores intentaban resolver problemas cada vez más complejos, las tareas que por su tamaño rebasaban la capacidad de los mejores programadores se convirtieron en norma. Así las cosas comenzaron a proliferar **equipos de programadores que trabajaban juntos** para emprender grandes tareas de programación.

Cuando ocurrió lo anterior se observó un fenómeno interesante: La misma tarea que un programador terminaría en dos meses no podían llevarla a cabo dos programadores en un mes. En palabras de **Fred Books:** *"Tener un bebe lleva nueve meses, sin importar cuántas mujeres participen en ello".* La razón de tal **comportamiento no lineal** era la **complejidad.**

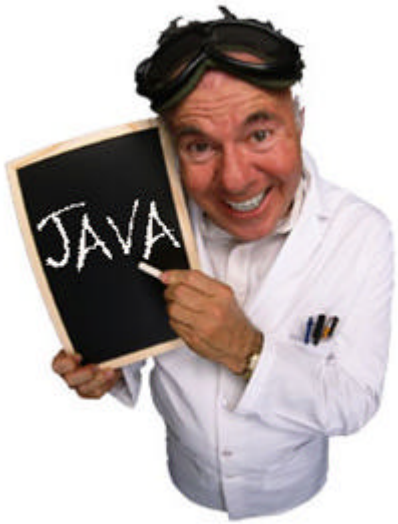
En la historia de la ciencia de la computación los programadores han tenido que lidiar durante mucho tiempo con el problema de la complejidad.

A fin de entender completamente la importancia de las técnicas orientadas a objetos, debemos revisar la variedad de mecanismos a los que han recurrido los programadores para controlar la complejidad.

A la cabeza de ellos está la **abstracción,** esto es, la **capacidad para encapsular y aislar la información de diseño y ejecución.**

En un sentido, las técnicas orientadas a objetos pueden verse como un producto natural de una larga progresión histórica que va de los procedimientos, a los **módulos,** los tipos de datos **abstractos** y, finalmente, los objetos.

5 Software reutilizable



Durante décadas, la gente ha preguntado por qué la construcción del software no refleja más de cerca la elaboración de otros objetos materiales.

ejemplo	Por ejemplo, cuando construimos un edificio, un automóvil o un aparato electrónico, por lo general se ensambla cierto número de componentes prefabricados, en vez de fabricar cada nuevo elemento partiendo de nada. **¿No podría el software construirse de la misma manera?**

Las técnicas orientadas a objetos proveen un mecanismo para separar con limpieza la **información esencial** de la **información de poca importancia.** De esta manera, usando las técnicas orientadas a objetos, podemos construir grandes componentes de software reutilizable.