

# Programación Orientada a Objetos

## Introducción a la programación orientadas a objetos.

La tecnología orientada a objetos se define como una metodología de diseño de software que modela las características de objetos reales o abstractos por medio del uso de clases y objetos[1]. Hoy en día, la orientación a objetos es fundamental en el desarrollo de software, sin embargo, esta tecnología no es nueva, sus orígenes se remontan a la década de los años sesenta. De hecho Simula, uno de los lenguajes de programación orientados a objetos más antiguos, fue desarrollado en 1967.

**Objeto: Una unidad de software conformada por atributos y métodos específicos.**

El objeto, es el concepto principal sobre el cual se fundamenta la tecnología orientada a objetos. Un objeto puede ser visto como una entidad que posee atributos y efectúa acciones. En el mundo real podemos encontrar cientos de ejemplos que cumplen con ésta definición, algunos de ellos son: una bicicleta, un automóvil, una persona, una computadora, etcétera.

Estos objetos son casos particulares de entidades llamadas clases en donde se definen las características comunes de tales objetos. Veamos el caso particular del objeto automóvil, podemos mencionar como atributos de éste: el modelo, el color, la marca, el número de placas, entre otros. Algunas acciones que es capaz de realizar un automóvil son: ir en reversa, virar, frenar, acelerar y cambiar la velocidad. Este objeto automóvil, es una instancia particular de la entidad automóvil. En términos de la programación orientada a objetos, se dice que todo objeto tiene un estado (atributos) y un comportamiento (acciones). La programación orientada a objetos nos permite modelar estos objetos del mundo real en objetos de software de forma eficaz. Un objeto de software mantiene sus atributos o estado en variables e implementa las acciones o comportamientos por medio de métodos o funciones. Supongamos que queremos desarrollar una aplicación que simule un vehículo. Tendríamos entonces un objeto vehículo constituido por

variables en las cuales podríamos almacenar número de serie, color, la velocidad actual, etcétera. Aunado a esto, tendríamos un conjunto de funciones que implementarían las acciones para frenar, virar, cambiar la velocidad, etcétera.

**Clase: Es un molde o bien prototipo en donde se definen los atributos (variables) y las acciones (métodos) comunes de una entidad.**

Este es el paradigma que propone la programación orientada a objetos, la abstracción de los elementos que constituyen a un objeto del mundo físico, esto es, atributos y comportamiento, y la representación de éstos elementos por medio de objetos de software formados por variables y métodos que permitan la manipulación de tales variables. Cabe mencionar que además de representar los objetos físicos del mundo real por medio de objetos, también podemos modelar objetos abstractos, por ejemplo las acciones de un usuario al interactuar con la máquina. Como observaremos más adelante, algunas de las ventajas de la tecnología orientada a objetos son la reutilización de objetos, y la facilidad de comprensión de los programas.

## **Bases sobre las cuales se fundamenta la programación orientada a objetos.**

Los elementos más importantes que deben tener los objetos de software, para cumplir con el paradigma de orientación a objetos son:

- Abstracción.
- Modularidad.
- Encapsulamiento.
- Jerarquía.
- Polimorfismo.

### **Abstracción.**

La abstracción es el proceso en el cual separamos las propiedades más importantes de un objeto, de las que no lo son. Es decir, por medio de la abstracción definimos las características esenciales de un objeto del mundo real, los atributos y comportamientos que lo definen como tal, para después modelarlo en un objeto de software.

En el proceso de abstracción no debemos preocuparnos por la implementación de cada método o atributo, solamente debemos definirlo de forma general. Por ejemplo, supongamos que deseamos escribir un programa para representar el sistema solar, por medio de la abstracción, podemos ver a éste sistema como un conjunto de objetos, algunos de estos objetos son los planetas, que tienen un estado, dado por sus características físicas, digamos, tamaño, masa, entre otras, estos objetos tendrán también comportamientos : la translación y la rotación, pueden mencionarse como los más evidentes.

Visualizar las entidades que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un diseño óptimo de nuestro software, ya que nos permite comprender más fácilmente la programación requerida.

En la tecnología orientada a objetos la herramienta principal para soportar la abstracción es la clase. Podemos definir a una clase como una descripción genérica de un grupo de objetos que comparten características comunes, dichas características se especificadas en sus atributos y comportamientos. En otras palabras, una clase es un molde o modelo en donde se especifican las características que definen a un objeto de manera general, a partir de una clase pedemos definir objetos particulares. En programación orientada a objetos se dice que un objeto es una instancia de la clase, es decir un objeto, es un caso particular del conjunto de objetos que comparten características similares, definidas en la clase.

Un ejemplo servirá para clarificar estos conceptos. Definamos a la clase persona, esta clase tendrá ciertos atributos, por ejemplo edad, sexo y nombre entre otros. Las acciones o comportamientos que podemos definir para esta clase son, por ejemplo, respirar, caminar, comer, etcétera. Estas son, para nuestro caso, las características que definen a nuestra clase persona. Un objeto instanciado de está clase podría ser Juan Pérez, de sexo masculino y de 35 años de edad, quién se encuentra caminando. Estos atributos son conocidos formalmente en programación orientada a objetos como variables de instancia por que contienen el estado de un objeto particular en un momento dado, en este caso Juan Pérez. Así mismo, los comportamientos son llamados métodos de instancia porque nos muestran el comportamiento de un objeto particular de la clase.

**Hay que tener cuidado de no confundir un objeto con una clase, una gelatina de fresa y una gelatina de limón son objetos de la misma clase (o del mismo molde), pero con atributos (o sabores) diferentes.**

## **Modularidad.**

Dentro de la programación orientada a objetos, la modularidad juega un papel muy importante. Una vez que hemos representado una situación del mundo real en un programa, tenemos regularmente como resultado, un conjunto de objetos de software que constituyen la aplicación. La modularidad, nos permite poder modificar las características de la clase que definen a un objeto, de forma independiente de las demás clases en la aplicación. En otras palabras, si nuestra aplicación puede dividirse en módulos separados, normalmente clases, y estos módulos pueden compilarse y modificarse sin afectar a los demás, entonces dicha aplicación ha sido implementada en un lenguaje de programación que soporta la modularidad. La tecnología orientada a objetos nos brinda esta propiedad para hacer uso de ella en el software que desarrollemos.

## **Encapsulamiento.**

También referido como ocultamiento de la información, el encapsulamiento es la propiedad de la orientación a objetos que nos permite asegurar que la información de un objeto le es desconocida a los demás objetos en la aplicación. Es muy frecuente referirse a los objetos de software como cajas negras, esto se debe principalmente a que no necesitamos, dentro de la programación orientada a objetos, saber como esta instrumentado un objeto para que este interactúe con los demás objetos. Generalmente una clase se define en dos partes, una interfaz por medio de la cual los objetos que son instanciados de la misma interactúan con los demás objetos en la aplicación, y la implementación de los miembros de dicha clase (métodos y atributos). Retomando dos de los ejemplos anteriores, supongamos que deseamos realizar una aplicación en donde deben interactuar el objeto Mercedes Benz, instancia de la clase automóvil y el objeto Juan Pérez, instancia de la clase Persona. Ambos objetos pueden ser vistos como cajas negras las cuales se comunican por medio de una interfaz. El objeto Mercedes no sabe como esta implementado el objeto Juan Pérez y viceversa. Concretamente, el encapsulamiento permite a un objeto ocultar información al mundo exterior, o bien restringir el acceso a la misma.

Una aplicación orientada a objetos esta constituida, como mencionamos anteriormente, por módulos. Estos módulos se implementan mediante clases, las cuales representan, generalmente, abstracciones de objetos del mundo real. Es por medio del encapsulamiento que podemos definir los atributos y los métodos de una clase para que los objetos que se instancian de ésta trabajen como unidades independientes de los demás objetos con los que interactúan. En otras palabras, con el encapsulamiento ganamos modularidad, y además protegemos a los objetos de ser manipulados de forma inadecuada por objetos externos.

## **Jerarquía**

Una vez que hemos definido una clase, por ejemplo la clase bicicleta, con sus atributos y métodos, tal vez necesitemos definir una nueva clase específica para las bicicletas de carreras. Es obvio que ésta nueva clases compartirá elementos en común con la clase bicicleta, es decir, la clase bicicleta de carreras será un subconjunto de la clase bicicleta.

La tecnología orientada a objetos nos permite definir jerarquías entre clases y jerarquías entre objetos. Las dos jerarquías más importantes que existen son la jerarquía “es un” que precisa la generalización y especificación entre clases y la jerarquía “es parte de” en la cual se delimita la agregación de objetos.

Comúnmente, a la jerarquía “es un” se le conoce como herencia. La herencia simple es la propiedad que nos permite definir una clase nueva en términos de una clase ya existente. Regresando al ejemplo, si podemos decir que una bicicleta de carreras “es una” bicicleta, entonces podemos definir a la clase bicicleta de carreras a partir de la clase bicicleta. Existirán casos en los cuales se necesite definir una clase a partir de dos o más clases preexistentes, en este caso estaremos hablando de herencia múltiple.

En cuanto a la jerarquía de agregación, también conocida como inclusión, podemos decir que se trata del agrupamiento lógico de objetos relacionados entre sí dentro de una clase.

Supongamos que tenemos que definir a la clase automóvil, pero además hemos definido ya a la clase volante. Si podemos expresar la oración, un volante “es parte de” un automóvil, entonces podemos instanciar un objeto de la clase volante, para definir a la clase automóvil. En este caso, se dice que automóvil es una agregación y volante es un agregado de automóvil.

## Polimorfismo


Muchas veces es necesario que un mismo comportamiento o acción se realice de diferentes maneras, por ejemplo, supongamos que deseamos implementar a la clase mamíferos, supongamos también que uno de los métodos que deseamos implementar para esta clase, es el que permita a tales mamíferos desplazarse de forma natural. Nos encontraremos entonces con que para algunos mamíferos el desplazamiento se realizará por medio de caminar, como es en el caso de las personas, para otros el desplazamiento natural será nadar, como en el caso de los delfines e inclusive para otros, el desplazamiento se logrará por medio de volar, como sucede con los murciélagos. En otras palabras, un mismo comportamiento, en este caso el desplazamiento, puede tomar diferentes formas.

Dentro de la programación orientada a objetos puede modelarse esta situación del mundo real, en objetos de software, gracias al polimorfismo. El polimorfismo es la propiedad por la cual una entidad puede tomar diferentes formas. Generalmente esta entidad es una clase, y la forma en que se consigue que tome diferentes formas es por medio de nombrar a los métodos de dicha clase con un mismo nombre pero con diferentes implementaciones.

## Taxonomía de los lenguajes orientados a objetos.

Una clasificación para los lenguajes de programación con respecto a la orientación a objetos fue creada por Wegner[2], dicha clasificación se constituye de la siguiente forma:

Basados en objetos	Basados en clases	Orientación a Objetos
Si la sintaxis y semántica de lenguaje soportan las características de objetos que hemos mencionado.	Si un lenguaje esta basado en objetos y además soporta clases, se dice que esta basado en clases.	Si un lenguaje de programación soporta objetos, clases y además permite la jerarquía de dichas clases, entonces se dice que es un lenguaje de programación orientado a objetos.



Esta clasificación es la que prevalece actualmente, en esta se establece claramente que para que un lenguaje de programación sea considerado orientado a objetos éste debe soportar la creación de clases y la herencia.

Java es actualmente uno de los lenguajes de programación, que cumple perfectamente con los requisitos de la orientación a objetos, Delphi, Simula, Smalltalk, son también considerados lenguajes orientados a objetos.

Java puede ser visto como el resultado de una mezcla de C++ y Smalltalk, tiene la sintaxis de C++, lo que lo hace relativamente fácil de aprender, si es que se está familiarizado con C++, una de las características que lo hace más robusto que C++, es que no tiene apuntadores, que son causa común de errores en el desarrollo de aplicaciones con este lenguaje. Como en el caso de Smalltalk, Java tiene recolección de basura, una capacidad que libera al programador de definir funciones de alojamiento y desalojamiento de estructuras en memoria. Más adelante hablaremos de Java dentro del contexto de orientación a objetos con más detalle.

## **Características adicionales de los lenguajes orientados a objetos.**

Además de las características que mencionamos anteriormente como esenciales de los lenguajes de programación orientados a objetos, es deseable que éstos cumplan también con las siguientes:

**Tipificación fuerte.** Esto es, que durante la fase de diseño e implementación se declare que tipo de datos soportara cada variable.

**Manejo de excepciones.** Dentro de la misma definición del lenguaje se deberá establecer la forma de detectar y manipular excepciones que puedan surgir durante la ejecución de un programa.

**Paso de mensajes.** Es conveniente que el lenguaje soporte paso de mensajes entre módulos de manera bidireccional.

**Generalidad.** Se refiere principalmente a que las clases se definan lo más generalizadas posible para que sean fácilmente reusables. Para generar este tipo de clases, normalmente se definen parámetros formales que son instanciados por parámetros reales.

**Multitarea.** Es conveniente que el lenguaje permita la creación de procesos que se ejecuten de forma simultánea independientemente del sistema operativo.

**Persistencia.** Los objetos deben poder permanecer, si así se desea, después de la ejecución de un programa.

**Datos compartidos.** Los objetos pueden necesitar referirse a la misma localidad de memoria (memoria compartida) o bien comunicarse mediante mensajes.

## **Ventajas de la tecnología orientada a objetos.**

**Flexibilidad.** Si partimos del hecho que mediante la definición de clases establecemos módulos independientes, a partir de los cuales podemos definir nuevas clases, entonces podemos pensar en estos módulos como bloques con los cuales podemos construir diferentes programas.

**Reusabilidad.** Una vez que hemos definido a la entidad persona para utilizarla en una aplicación de negocios, por mencionar un ejemplo, y deseamos construir a continuación una aplicación, digamos de deportes, en donde requerimos definir a la misma entidad persona, no es deseable volver a escribir la definición para la entidad persona. Por medio de la reusabilidad podemos utilizar una clase definida previamente en las aplicaciones que nos sea conveniente. Es claro que la flexibilidad con la que se definió la clase va a ser fundamental para su reutilización.

**Mantenibilidad.** Las clases que conforman una aplicación, vistas como módulos independientes entre sí, son fáciles de mantener sin afectar a los demás componentes de la aplicación.

**Extensibilidad.** Gracias a la modularidad y a la herencia una aplicación diseñada bajo el paradigma de la orientación a objetos puede ser fácilmente extensible para cubrir necesidades de crecimiento de la aplicación.

## **Desventajas de la tecnología orientada a objetos.**

A pesar de que las ventajas de la programación orientada a objetos superan a las limitaciones de la misma, podemos encontrar algunas características no deseables en ésta.

**Limitaciones para el programador.** No obstante que la tecnología orientada a objetos no es nueva, un gran porcentaje de programadores no están familiarizados con los conceptos de dicha tecnología. En otras palabras, la lógica de la programación estructurada sigue siendo predominante en la mayoría de los desarrolladores de software, después de haber revisado de forma breve los principios de la programación orientada a objetos, nos es claro que en ésta se requiere una lógica de pensamiento totalmente diferente a la lógica comúnmente utilizada para la programación estructurada.

**Tamaño excesivo en las aplicaciones resultantes.** La gran mayoría de los equipos de computo cuentan con capacidades tanto de almacenamiento como de memoria lo suficientemente buena como para ejecutar la mayoría de las aplicaciones que puedan desarrollarse con la tecnología orientada a objetos, sin embargo existen casos en los que lo anterior no se cumple. Una de las desventajas de la programación orientada a objetos es que cuando se heredan clases a partir de clases existentes se heredan de forma implícita todos los miembros de dicha clase aun cuando no todos se necesiten, lo que produce aplicaciones muy grandes que no siempre encajan en los sistemas con los que se disponga.

**Velocidad de ejecución.** Esto tiene que ver, en cierto modo, con el punto anterior, una aplicación innecesariamente pesada en muchas ocasiones es más lenta de ejecutar que una aplicación conformada únicamente por los módulos necesarios.

## **Resumen.**

A manera de conclusión podemos decir que la tecnología orientada a objetos nos permite diseñar e implementar sistemas bajo un paradigma de programación en el cual, por medio de la abstracción definimos a las clases o entidades de software a partir de entidades del mundo real. De dichas entidades instanciamos objetos de software que se corresponderán con los objetos reales que representan. Esto nos permite enfocarnos más en la solución del problema que en la implantación de dicha solución.

Ledbetter y Cox[3] lo establecen de la siguiente manera:

**La programación orientada a objetos permite una representación más directa del modelo del mundo real en el código. El resultado es que la radicalmente transformación normalmente llevada a cabo de los requisitos del sistema (definido en términos de usuario) a la especificación del sistema (definida en términos de computadora) se reduce considerablemente.**

[1] The Java Tutorial.

[2] Joyanes, Programación orientada a objetos, pág. 28, Mc Grow Hill 1998.

[3] Luis Joyanes A. Programación Orientada a Objetos, página 18. Mc Grow Hill, 1998.

**José Arturo de los Angeles** es pasante de la Lic. en Ciencias Computacionales de la Benemérita Universidad Autónoma de Puebla (México).

Para cualquier duda o tirón de orejas, e-mail a: [arturoas7@yahoo.com](mailto:arturoas7@yahoo.com)