Programación Orientada a Objetos

I.F.T.S. 4 D.E. 7

Alberto Blumberg

Daniel Fernández

Javier Oshiro

Guillermo Ranucci

SEP-2015

Índice

Programación Orientada a Objetos (POO)

Introducción

La programación orientada a objetos (POO) es un modelo de programación con su teoría y metodología, un lenguaje orientado a objetos es el que permite el diseño de las aplicaciones con estas metodologías, por lo que es conveniente primero aprender el método para después aplicarlo en los lenguajes, porque el método es uno solo, pero los lenguajes son muy variados.
La programación orientada a objetos es una nueva forma de pensar acerca de la descomposición de los problemas y el desarrollo de soluciones, no se trata de hacer más sencillo el problema a resolver dividiéndolo en partes, sino que se trata de simular el problema. Lo que se simula son entidades físicas o no que interaccionan en la resolución, de modo que simulando estas y su comportamiento es posible llegar a una solución. Cada una de estas entidades es un objeto.

Los objetos son entidades que tienen ciertas características y propiedades:
- Estado: datos e información asignada al objeto.
- Comportamiento: operaciones que puede realizar.
- Identidad: propiedad que lo diferencia del resto.

Un objeto contiene la información que permite identificarlo y diferenciarlo de otros objetos tanto de la misma clase como de otra totalmente diferente, estos interactúan mediante métodos que alteran el estado de los propios objetos haciendo que el comportamiento y el estado estén directamente relacionados, esta relación se conoce como la propiedad de conjunto. Esta dicta que una clase requiere a los métodos para poder modificar los atributos propios.

Este paradigma de programación se diferencia de la programación estructurada tradicional en que los datos y los procedimientos están juntos y relacionados, y se piensa primero en definir los objetos para luego enviarles mensajes para que realicen sus métodos.

Origen

Conceptualmente la programación orientada a objetos tiene origen en Noruega en el lenguaje Simula 67 (1967), este estaba diseñado para realizar simulaciones de naves con la idea de que las características de las naves podían afectar unas a otras, agrupando las distintas naves en clases de objetos con la capacidad de definir sus propios datos y comportamiento.

El problema durante los años '70 fue que los sistemas no se terminaban con los requerimientos iniciales y los que si se terminaban no se usaban según lo planificado. El problema era cómo adaptar el software a requerimientos que no se pueden planificar inicialmente, ya que el hombre crea y aprende en base a la experimentación y no a la planificación. La orientación a objetos permite experimentar y escribir código sin tener todo planificado.

Así surge Small Talk de Xerox que fue el primer lenguaje puro de orientación a objetos, cuando D. Parnas propone ocultar información encapsulando las variables globales en un solo módulo junto con sus operaciones asociadas que eran la única forma de acceder a esas variables, y el resto de los módulos podían tener acceso a estas de forma indirecta mediante operaciones específicas.

Más tarde en 1980, Stroustrup de AT&T Labs amplía el lenguaje C a C++ agregándole soporte a la programación orientada a objetos convirtiéndolo en el estilo dominante con C++ como el más influyente, su consolidación se dio gracias a la aparición de las interfaces gráficas para las cuales se usaba este paradigma, y a la reducción de costos en hardware que significó mayor uso de la informática.

En 1996 surge JAVA como extensión de C++ con la idea de aprovechar el software ya existente, utilizándolo para otros usos diferentes a los originales sin modificar el código.
Las características de la programación orientada a objetos fueron agregadas a lo largo del tiempo en muchos lenguajes ya existentes no diseñados para este tipo en particular, por lo que produjo problemas de compatibilidad y mantenimiento de código. Para resolverlo se crearon nuevos lenguajes orientados a objetos pero permitiendo características de la programación tradicional.

Conceptos

- Clase

Es la definición de las propiedades (estructura de datos) y comportamiento (métodos) de un tipo de objeto, se le dice instancia a la creación de un objeto a partir de esta clase. Aunque objetos diferentes de una misma clase pueden tener atributos distintos, deben obligatoriamente tener el mismo comportamiento.
Por ejemplo: existen rectángulos de varios tamaños, todos ellos pertenecientes a una clase "rectángulo" que se define con atributos comunes para todos: base y altura, y una función calcularArea() también común para todos.

```
class rectángulo {
        private:
                int base;
                int altura;
        public:
                rectángulo(int b, int h) {
                        base =b;
                        altura=h;
                }
                float calcularArea() {
                        return base*altura;
                }
}
```

Una superclase es una clase de mayor nivel jerárquico que agrupa otras clases con sus propiedades y funciones, por lo que los objetos de la subclase (clase agrupada dentro de otra) son también objetos de la superclase.

- Herencia

Es una característica de las clases por la cual una puede heredar los atributos y operaciones de otra, de esta manera usan los mismos métodos y variables públicas de la primera.

- Objeto

Es una instancia de una clase, con sus propiedades y métodos.

- Método

Es un proceso asociado a un objeto, puede producir un cambio en las propiedades del objeto o un mensaje de evento para otro objeto diferente. Es una función que usa como variables locales de las propiedades del objeto.

- Mensaje

Los objetos se comunican a través de mensajes que le dicen al objeto lo que tiene que hacer, señalando el método y los parámetros necesarios para la ejecución.
Los mensajes consisten en una dirección, un método y los parámetros si es que hay: OBJETO.método(parámetros).
Por ejemplo: del caso anterior de los rectángulos teniendo un objeto rectángulo llamado "rect1" podemos formular el mensaje.

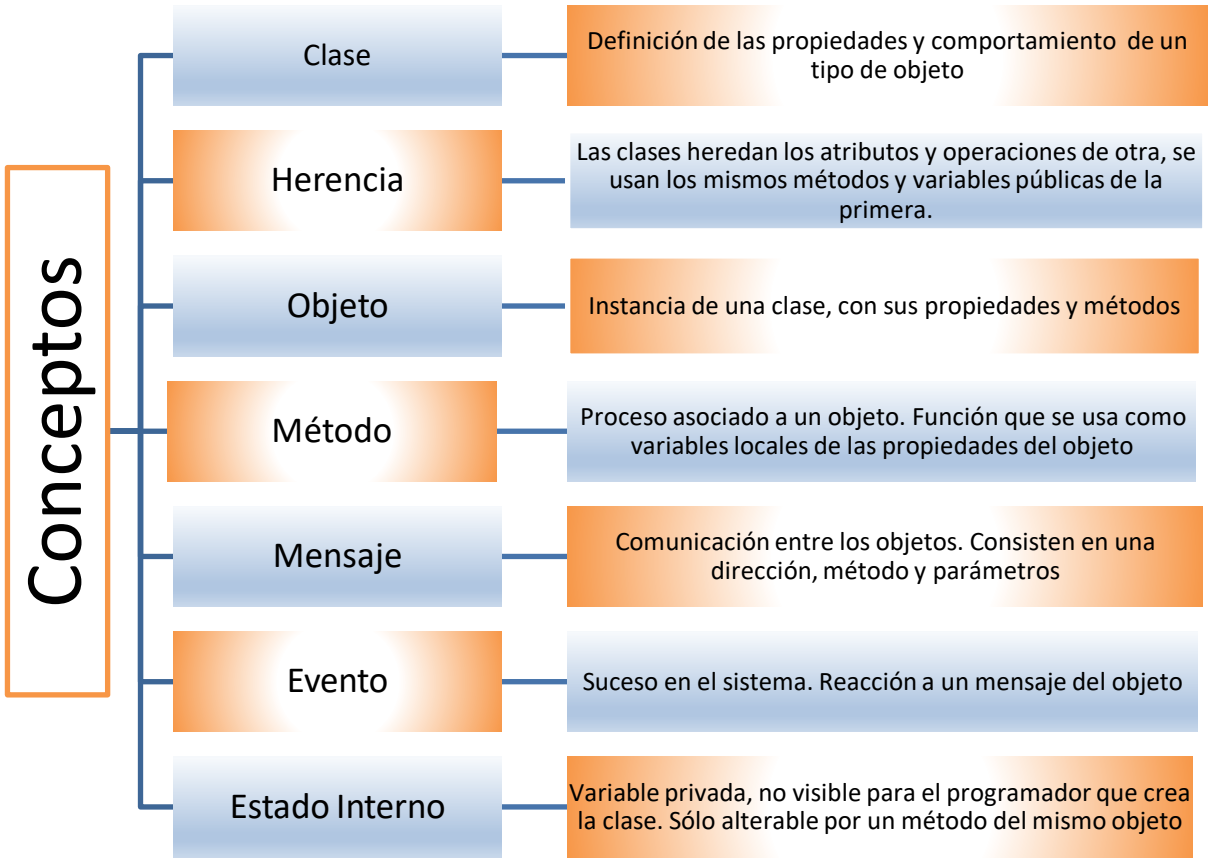
```
rect1.calcularArea();
```

- Evento

Es un suceso en el sistema, puede ser el usuario interactuando con la PC o un mensaje de otro objeto, se considera al evento como la reacción a un mensaje del objeto.

- Estado interno

Es una variable privada que solo puede ser alterada por un método del mismo objeto, no es visible para el programador que crea la instancia de la clase.



Características

- Abstracción

Tiene que ver con capturar el comportamiento y los atributos, extrayendo las características más destacadas para poder diferenciarlo de otros objetos. Definir una abstracción significa describir una entidad del mundo real y utilizarla en un programa. La abstracción nos ayuda a la hora de desarrollar una aplicación, permitiéndonos identificar los objetos que van a formar parte de nuestro programa, sin necesidad de disponer aún de su implementación, nos basta con reconocer los aspectos conceptuales que cada objeto debe resolver.
Cuando hablamos de una clase abstracta nos referimos a que si su aspecto interno es modificado, las otras partes del sistema seguirán funcionando sin verse afectadas. Un método abstracto es cuando se declara una funcionalidad que debe ser implementada en todas las subclases teniendo cada subclase su propia implementación del método.

- Encapsulamiento

Tiene que ver con aislar al objeto de manera que solo pueda ser modificado por sí mismo. Agrupando los datos como atributos de un objeto y los métodos dentro de las clases.
Proporciona seguridad al código de la clase, evitando accesos y modificaciones no deseadas, una clase bien encapsulada no debe permitir la modificación directa de una variable, ni ejecutar métodos que sean de uso interno para la clase. También facilita el uso ya que un programador no necesita conocer los detalles de sus métodos, simplemente los usa.

- Ocultamiento

Se usa para prevenir cambios en la implementación del objeto y de esta forma otros programas que usaban este mismo objeto no sufran alteraciones. La organización de la clase es invisible desde el exterior ya que se establecen como privadas, en cambio los métodos públicos que conforman la interfaz del objeto son visibles. Esto no quiere decir que la información del objeto no se pueda conocer, sino que para obtener información hay que hacerlo a través de mensajes.

- Modularidad

Tiene que ver con separar una aplicación en partes más chicas y sencillas cada una independiente de la aplicación y de otros módulos.

- Polimorfismo

Permite que un método pueda ser utilizado tanto por la clase a la que pertenece como por las clases que heredan el comportamiento de la primera y que la operación varíe según el objeto que lo aplique, siendo el mismo método. Puede traer problemas al usar el mismo nombre para el método siendo la operación diferente.

- Herencia

Es quizá la característica más importante de este paradigma, permite crear objetos especializados de objetos ya existentes, compartiendo el comportamiento. Es la transmisión de código entre las diferentes clases, hay una clase padre y otra clase hija, la primera transmite a la segunda el código en su totalidad ahorrando el tener que copiarlo. Desde el punto de vista de implementación se trata de crear nuevas clases a partir de otras existentes, reutilizando el comportamiento para definir otra nueva que hereda los atributos y métodos de su superclase, pudiendo las subclases especializarse añadiendo atributos y métodos adicionales o cambiando los heredados.

La herencia puede ser:

-Simple: cuando la clase que recibe la herencia tiene un solo padre.

-Múltiple: cuando la clase que recibe la herencia tiene varios padres o alguno de sus antepasados tiene varios padres. Permite definir clases hibridas que comparten el comportamiento de dos o más clases, lo que suele traer conflictos en los nombres de los atributos o métodos.

Otra clasificación de la herencia es:

-Selectiva: se eligen qué métodos se heredan.

-No selectiva: se heredan todos los métodos y atributos.

- Jerarquización

Es el orden o clasificación de los objetos, puede ser por composición teniendo un objeto que está compuesto por muchos otros, o por clasificación cuando un objeto es una especialización de otro por herencia. Teniendo en cuenta la herencia podemos organizar las clases jerárquicamente:

-Raíz o superclase: es una clase que suele ser única al estar situada en el punto más alto de la escala jerárquica.

-Clases intermedias: clases que descienden directa o indirectamente de la raíz y tienen a su vez descendientes, se les llama clase abstracta cuando descienden de clases intermedias y clases de objetos terminales cuando descienden de una clase terminal.

-Clases terminales: clases que descienden de otra y no tienen descendientes, los objetos normalmente se instancian de estas clases.

- Asociación

Las relaciones de asociación entre clases pueden ser de dos tipos:

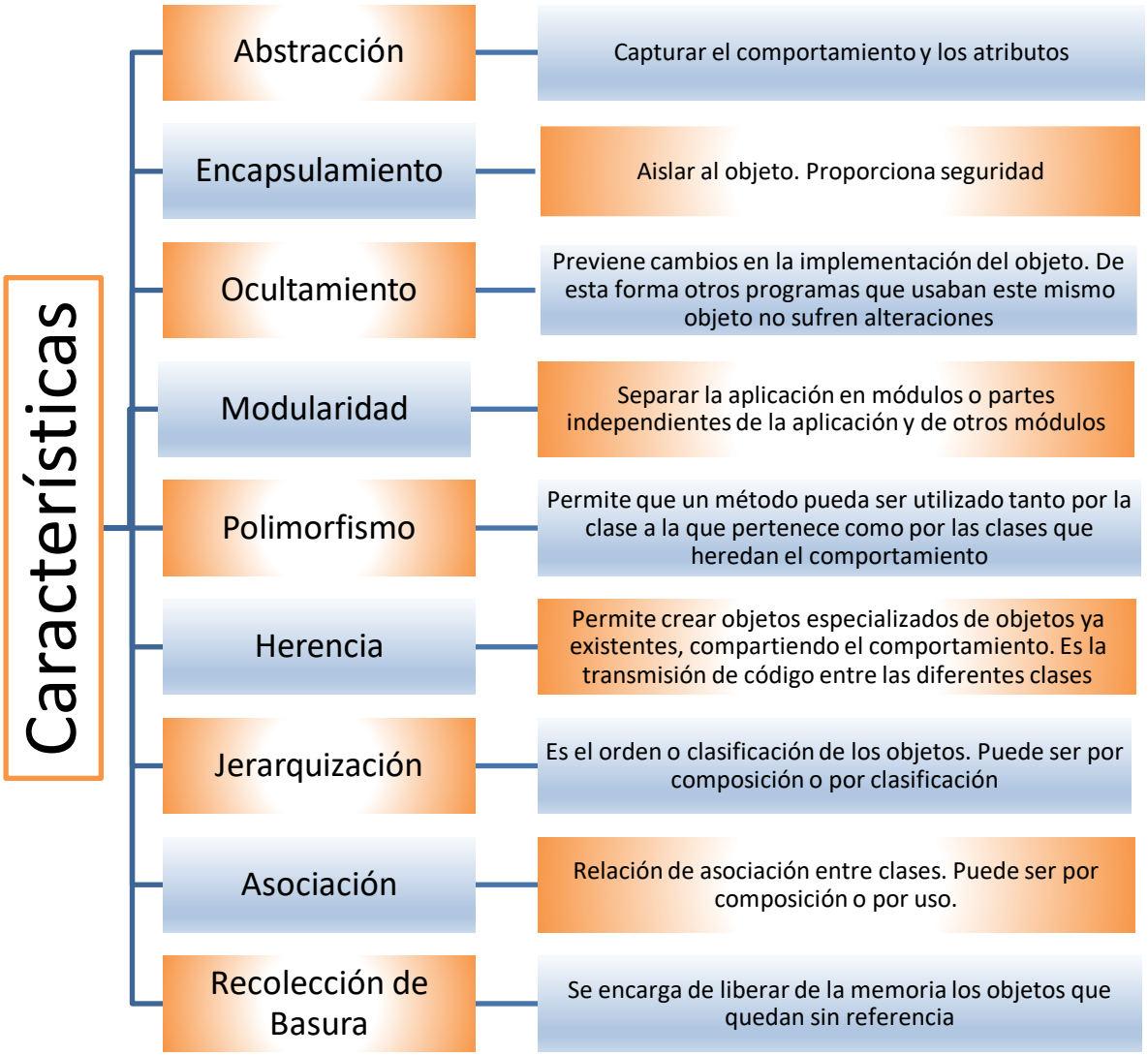
-Composición: cuando una clase es parte de otra.

-Uso: cuando una clase usa a otra sin necesidad de que esta forme parte de la primera.

- Recolección de basura

Es una técnica que se encarga de liberar de la memoria los objetos que quedan sin referencia, esto significa que automáticamente se asigna memoria cuando se crea un objeto y se libera cuando ya no se esté usando. En la mayoría de los lenguajes híbridos que se

modificaron para usar este paradigma no existe este proceso por lo que la memoria debe ser desasignada manualmente.



Completando el ejemplo anterior con algunos nuevos conceptos, definimos la clase figura con un método.

```
class figura {
        public:
                float calcularArea();
}
```

Defino la clase rectángulo que hereda de figura.

```
class rectángulo : public figura {
        private:
                float base;
                float altura;
        public:
                rectángulo(float b, float h) {
                        base = b;
                        altura=h;
                }
                float calcularArea() {
                        return base*altura;
                }
}
```

Defino la clase círculo que hereda de figura.

```
class círculo : public figura {
        private:
                float radio;
        public:
                círculo(float r) {
                        radio=r;
                }
                float calcularArea() {
                        return radio^2*3.14;
                }
}
```

UML (Unified Modeling Language)

Es el lenguaje de modelado de sistemas más conocido y utilizado en la actualidad. Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.
El diagrama de clases es una herramienta para comunicar el diseño de un programa orientado a objetos, permitiendo modelar las relaciones entre las entidades. En UML, una clase es representada por un rectángulo que posee tres divisiones: Nombre de la clase, atributos que tiene y mensajes que entiende.
- En el primer cuadro anotamos el nombre de la clase (si es abstracta se escribe en cursiva, o bien se usa un estereotipo <<abstract>> arriba del nombre de la clase).
- En la segunda parte van los atributos (o variables de instancia, las variables de clase van en subrayado).
- En el último cuadro escribimos las operaciones (qué mensajes que puede entender). Lo importante no es documentar todos los mensajes de un objeto, sino sólo los más relevantes.
En el diagrama las relaciones entre los objetos y las clases se establecen mediante flechas y líneas, usando distintas formas para cada tipo de relación.

Impacto del Paradigma Orientado a Objetos

- Reusabilidad de código: Hasta el momento para poder reutilizar código se tenía que usar el 100% del mismo lo que complicaba el programa y dificultaba su mantenimiento, ahora se permite reutilizar un diseño diferente del que se pensó originalmente, y si una aplicación tiene partes que están adecuadas a nuestras necesidades podemos mediante herencia corregirlo.
Fiabilidad y mantenimiento del código: al tener librerías de clases y habiendo asegurado que esas clases tienen verificado su funcionamiento garantiza la fiabilidad, lo que hace que una rutina este compuesta por unas pocas líneas de código.

Ventajas del Paradigma Orientado a Objetos

La primera ventaja del concepto de objetos es que todo el código que tiene algo que ver con un objeto en particular se encuentra en un solo lugar.
Otra ventaja es que los objetos pueden poseer atributos inherentes de la clase a la que pertenecen, por ejemplo, naves espaciales y asteroides podrían tener ambos una posición XY porque todos los objetos que pertenecen a la clase de los objetos en movimiento tiene una posición XY. Escribir códigos es más fácil porque se pueden conceptualizar como algo que le sucede a un objeto.
Otra ventaja es que el paradigma hace que los programas grandes sean más manejables. Si todas las ventanas pertenecen a una jerarquía de clases de ventanas y todo el código que se refiere a una ventana particular está dentro de esa ventana, todas las manipulaciones de ventana se pueden escribir como una sencilla transferencia de mensajes.

Desventajas del Paradigma Orientado a Objetos

No todos los programas pueden ser modelados con exactitud por el modelo de objetos. Si lo que se requiere es leer algunos datos, hacerles algo simple y escribir de nuevo, no hay necesidad de definir clases y objetos. Sin embargo, en algunos lenguajes de objetos, puede que sea obligatorio definirlos.
Otra desventaja es que si se fuerza el lenguaje en el concepto de programación orientada a objetos, se pierden algunas de las características de lenguajes útiles, como los "lenguajes funcionales".
Otra desventaja es que el concepto que un programador tiene de lo que constituye un objeto abstracto puede no coincidir con la visión de otro programador. Los objetos a menudo requieren una extensa documentación.
Por otro lado la aplicación resultante cuenta con un tamaño muy grande ya que cuando se heredan clases también se heredan todos los miembros de la clase incluso cuando no son necesarios para el programa, de esta manera también se ve perjudicada la velocidad de ejecución.

Algunos lenguajes orientados a objetos

Simula (1967) es aceptado como el primer lenguaje que posee las características principales de un lenguaje orientado a objetos. Fue creado para hacer programas de simulación, en donde los "objetos" son la representación de la información más importante.
Smalltalk (1972 a 1980) es posiblemente el ejemplo canónico, y con el que gran parte de la teoría de la programación orientada a objetos se ha desarrollado.
Entre los lenguajes orientados a objetos se destacan los siguientes:

ABAP1	Oz
ABL2	R
ActionScript	Pauscal (en español)
ActionScript 3	Perl6 7
Ada	PHP8
C++	PowerBuilder
C Sharp (C#)	Python
Clarion	Ruby
Object Pascal (Embarcadero Delphi)	Self
Gambas	Smalltalk9
GObject	Magik (SmallWorld)
Genie	Vala
Harbour	VB.NET
Eiffel	Visual FoxPro10
Fortran 90/95	Visual Basic 6.0
Java	Visual DataFlex
JavaScript4	Visual Objects
Lexico5	XBase++
Objective-C	DRP
Ocaml	Scala11 12

Muchos de estos lenguajes de programación no son puramente orientados a objetos, sino que son híbridos que combinan la POO con otros paradigmas.
Al igual que C++, otros lenguajes, como OOCOBOL, OOLisp, OOProlog y Object REXX, han sido creados añadiendo extensiones orientadas a objetos a un lenguaje de programación clásico.
Un nuevo paso en la abstracción de paradigmas de programación es la Programación Orientada a Aspectos (POA). Aunque es todavía una metodología en estado de maduración, cada vez atrae a más investigadores e incluso proyectos comerciales en todo el mundo.

Bibliografía

Wikipedia - https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

http://www.desarrolloweb.com/manuales/teoria-programacion-orientada-objetos.html
http://codejavu.blogspot.com.ar/2013/05/conceptos-de-programacion-orientada.html
http://sis324loo.blogspot.com.ar/2008/09/historia-de-los-lenguajes-de_29.html

Introducción a la Programación Orientada a Objetos – Luis R. Izquierdo

Introducción a la OOP – Francisco Morero (1999-2000) Grupo EIDOS

Programación Orientada a Objetos – Roberto Rodriguez Echeverría / Encarna Sosa Sánchez/ Alvaro Prieto Ramos.

Programación Orientada a Objetos – Luis Joyanes Aguilar

Aprendiendo Java y POO – Gustavo Guillermo Pérez

Programación en Visual Basic. Net – Luis Miguel Blanco (Grupo Eidos)