Unidad 2

Programación orientada a objetos

Programación

Índice

Introducción

La programación orientada a objetos (POO), de la misma manera que la programación estructurada, nace del paradigma de programación imperativa. Nace del lenguaje de programación Simula 67, que incluyó por primera vez en el año 1962 los conceptos de objeto y clase.

Este capítulo contemplará el paradigma de programación orientada a objetos, veremos los elementos de los que se compone, así como algunos conceptos de diseño utilizando el lenguaje unificado del modelado (UML), para agilizar la comprensión y ofrecer ejemplos visuales.

Al finalizar esta unidad

- Definiremos el paradigma de programación orientada a objetos.
- Conoceremos los elementos que se emplean en programación orientada a objetos.
- Comprenderemos la simbología básica de UML para representar objetos y sus relaciones.
- Nos familiarizaremos con los fundamentos de la programación orientada a objetos.

2.1. Auge de la programación orientada a objetos

Debido a las ventajas que proporciona la POO frente a otros paradigmas, desde la década de los ochenta es el paradigma de programación más utilizado.

- En primer lugar, posibilita, más intuitivamente que otros paradigmas, analizar los dilemas a resolver. En la POO, código y datos están entrelazados, mientras que, en la programación estructurada, estos se encuentran separados. La interacción entre distintos objetos permite acercarnos a un modelo que simule el mundo real de la mejor manera. Esto supone un cambio en el diseño del código y el enfoque del desarrollo, sobre todo con respecto a la programación estructurada.
- Debido a que la POO permite crear programas escalables y modulares, hace que abordar programas de gran tamaño sea más sencillo que con la programación estructurada.
- La POO también presenta una interfaz gráfica más depurada, de manera que la facilita su comprensión al usuario. Además, facilita el trabajo a los programadores, gracias a que podemos reutilizar código sin tener que escribirlo nuevamente.

2.2. Características de la programación orientada a objetos

El paradigma de POO supuso una revolución en el mundo de la programación al incluir novedosos conceptos que permitían generar código de más calidad, facilitar el desarrollo y reutilizar de una forma más práctica el código desarrollado.

De la misma forma que la POO partió de paradigmas de programación existentes, otros paradigmas han tomado la POO como punto de partida para ser desarrollados. Con la creciente demanda de aplicaciones que ofreciesen una interfaz gráfica cada vez más sofisticada, surgió la necesidad de adaptar la POO a un tipo de programación que permitiese controlar los eventos que se producían sobre las interfaces gráficas con las que el usuario interactuaba.

A continuación, serán descritos cada uno de los elementos y características de este extendido paradigma.

Un código de mayor calidad, facilidad para el desarrollo y poder reutilizar código, hicieron del paradigma de POO, un antes y un después para la programación.

POO fue tomado como referencia a la hora de desarrollar otros paradigmas, al igual que hizo él mismo con otros. La necesidad de una interfaz gráfica más detallada hizo que la POO evolucionara para poder detectar cambios en la interfaz gráfica utilizada por los usuarios.

Ahora veremos las características y elementos.

2.2.1. Clase

La clase es el elemento principal de la POO. En la clase queda descrito el posible conjunto de datos (propiedades, campos o incluso ambos, dependiendo de que lenguaje de programación utilicemos) y el conjunto de operaciones que se realizarán sobre estos datos (métodos), los cuáles pueden ser constructores o destructores. A todos estos elementos se les conoce como miembros.

De una clase podemos obtener la cantidad de objetos instanciados que se requieran; por lo que una clase es considerada como una plantilla con la cual crear y definir objetos.

En UML las clases están representadas a través de diagramas de clase, que consisten en una caja dividida en 3 áreas donde se encuentran el nombre de la clase y de sus miembros:

- **Área superior:** se encuentra el nombre de la clase (Persona).
- **Área intermedia:** aquí se encuentran los campos de la clase (DNI, nombre, apellido 1, apellido2, fechaNacimiento).
- **Área inferior:** aquí encontramos los métodos de la clase (el método constructor Persona).

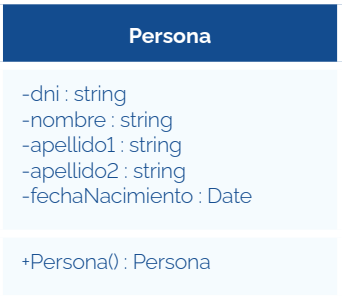

Imagen 1. Representación de una clase.

A parte de toda esta información, también podemos encontrar información respecto a la visibilidad de los diferentes miembros de una clase.

Campo

Los elementos que actúan como variables propias de cada instancia de una clase y que definen un atributo de un objeto, son llamados campo.

En el campo podemos almacenar datos, podemos inicializar, modifica y leer los datos de estos sin necesidad de una acción adicional. Poder o no leer los datos dependerá finalmente de la visibilidad del campo en cuestión.

Los datos pueden ser determinados de cualquier tipo incluyendo otras clases, y pueden ser inicializados al mismo tiempo que son declarados, o en su constructor.

Propiedad

Una propiedad, de la misma manera que un campo, determina un atributo de un objeto. Pero a diferencia de un campo que cuenta con un área de almacenamiento a la cual se puede acceder directamente, la propiedad no cuenta con esto. La propiedad permite el acceso a un área de almacenamiento mediante un campo definido con anterioridad o utilizando un método creado con ese objetivo. Constituye un sistema flexible e indirecto de acceso, con el que se puede establecer un conjunto de acciones, las cuales se llevaran a cabo de manera subyacente cada vez que se utilice para tener acceso a los datos, proporcionando así, un mejor control en el acceso.

Normalmente las propiedades se declaran con un nombre que represente el dato que van a contener, y los métodos utilizados para su modificación o lecturas se denominan set o get, respectivamente.

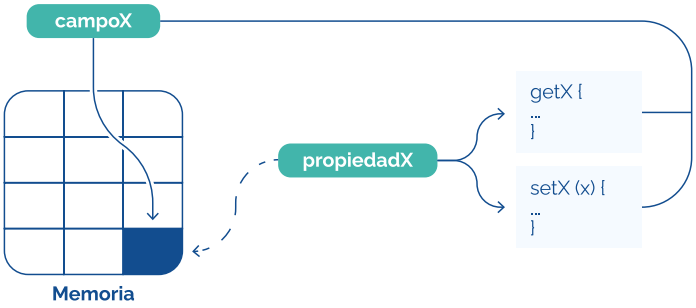

Imagen 2. Acceso mediante campos versus propiedades.

Dependiendo del tipo de lenguaje de programación utilizado, la forma de declarar las propiedades varía. Java, por ejemplo, no ofrece ningún mecanismo específico para su implementación.

Método

El método es la forma en que se implementa un algoritmo que se aplica a todas las instancias de una clase, y si se desea, puede utilizar los datos que esta contiene.

Un método tiene las siguientes formas:

- **Función.** Cuando el algoritmo que contiene, al ser llamado, realiza una serie de instrucciones y por último devuelve un valor.
- **Procedimiento.** Cuando el algoritmo que contiene, al ser llamado, lleva a cabo una serie de acciones, normalmente basadas en un conjunto de argumentos o parámetros. Pero en este caso no devuelve ningún valor al final.

Varios son los factores de los que depende que un método se implemente como procedimiento o como función. Primero, depende del lenguaje de programación utilizado, ya que algunos como Java solo utilizan funciones, y otros como Delphi distinguen entre funciones y procedimientos.

Constructor

Un método constructor es un tipo especial de método que pertenece a una clase concreta y utilizado para instanciar objetos que pertenecen a esta.

Cada clase puede contar con varios métodos constructores definidos, aunque también puede suceder el caso de que no se defina ninguno. Que no se defina un método constructor, no significa que no exista; todas las clases cuentan por defecto con un método constructor que, al llamarse, inicializará a los valores por defecto los campos de dicha clase.

Destructor

Un método destructor es aquel que se encarga de destruir un objeto y liberar aquellos recursos que este tuviera asignados.

Una clase puede tener un número indefinido de destructores, pero es posible que no se declare ninguno, al igual que los constructores. Aunque esto ocurra, no significa que no exista ninguno, ya que todas las clases cuentan con un método destructor por defecto.

Son los programadores los que tienen la responsabilidad de utilizar los métodos destructores cuando no existan mecanismos automáticos de destrucción de objetos.

2.3.2. Objeto

La instancia de una determinada clase se llama objeto. En otras palabras, podemos decir que un objeto resulta de la creación en tiempo de ejecución de un conjunto de datos, el cual es almacenado en memoria y que cuenta con una estructura como la descrita en su clase.

Las propiedades, campos y métodos que pertenecen a un objeto pueden declararse en una o varias clases. En caso de declararse en varias clases, heredará los campos de otra clase ancestra. Veremos con más detalle el concepto de herencia más adelante.

Entre los objetos encontramos los simples y los compuestos. Los simples están formados por datos primitivos (un dígito, un carácter, un booleano, etc.), mientras que los compuestos están formados por datos primitivos, y además por objetos pertenecientes a otras clases. Estudiaremos el concepto de composición de clases más adelante.

Dos objetos diferentes pueden tener los mismos datos, ya que cada uno tiene un espacio único e independiente en la memoria.

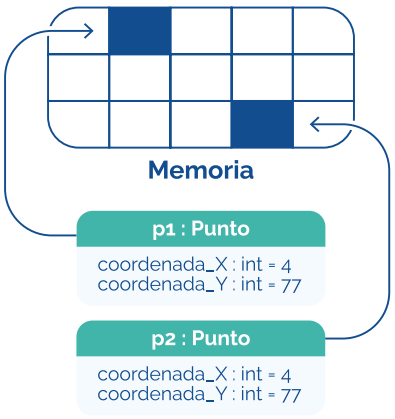

Imagen 3. Diferentes objetos con mismos valores.

2.3.3. Encapsulación

La encapsulación es la agrupación en un mismo elemento de los datos y las operaciones relacionadas con estos. Para que solo se pueda acceder mediante interfaces bien definidas, los datos encapsulados deben ser ocultados.

2.3.4. Abstracción

La abstracción es un mecanismo que permite reducir la complejidad de los objetos minimizando la información a representar, dejando solo los aspectos más relevantes. Hay que intentar obtener el nivel de abstracción necesario para que aspectos insustanciales no sean tomados en cuenta, ya que reducirían el rendimiento y dificultarían el desarrollo.

2.3.5. Reutilización o reusabilidad

Gracias a que el código ha sido escrito con anterioridad sin necesidad de duplicarlo, este modelo de programación permite reusar el código de forma sencilla. Es más, la reutilización del código es muy importante, ya que permite reducir la cantidad de líneas de código, reduce los errores a cometer, facilita la generación de programas y el mantenimiento, etc. Esta reutilización se consigue mediante técnicas de composición de objetos y de herencia.

2.3.6. Visibilidad

Tras la definición de encapsulación y abstracción, debemos definir el concepto de visibilidad. La visibilidad determina la manera en que los miembros de una clase son accedidos desde otros puntos del programa. Es importante no solo para crear relaciones de herencia entre distintas clases, sino también para implementar los algoritmos.

La visibilidad se puede aplicar a nivel de los atributos, métodos, clases y la forma de estructurar el programa, en paquetes y librerías.

Distinguimos entre los siguientes tipos de visibilidad:

- **Privada:** representada con el signo menos (-). Los miembros declarados con este tipo de visibilidad solo podrán verse desde la misma clase desde la que se declararon.
- **Pública:** representada con el signo de suma (+). Estos miembros podrán verse desde cualquier parte del programa.
- **Protegida:** representada con el signo de almohadilla (#). Estos miembros se verán solo desde la clase desde la que se declararon y aquellas clases que hereden de esta.

Existen tipos de lenguaje de programación que cuentan con algún tipo de visibilidad adicional.

2.3.7. Relaciones entre clases

Herencia

La herencia es una propiedad que solo poseen las clases que permite declarar nuevas clases a partir de otras clases declaradas con anterioridad. La clase original desde la que se hereda, se llama clase base, clase padre, clase ancestra o superclase, y la clase que hereda se llama clase extendida, clase hija, clase derivada o subclase.

Cuando una clase llamada "A" hereda de otra clase llamada "B", se transfieren desde la clase "A" a la "B" todos los miembros de esa clase, excepto destructores y constructores. La clase "B" puede añadir miembros de su propia clase (campos, métodos y propiedades) o puede sobrescribir los miembros que ha heredado.

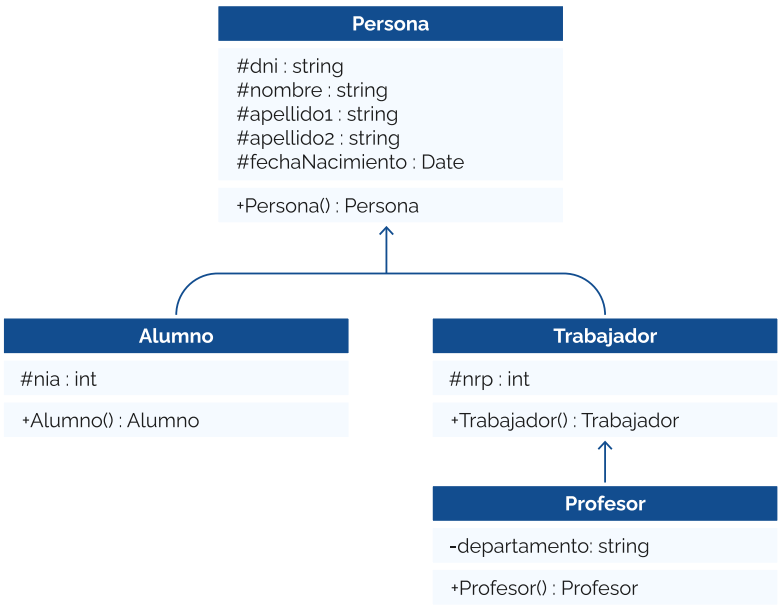

Imagen 4. Ejemplo de representación de la herencia.

El paradigma de la POO indica que la herencia puede ser múltiple, esto quiere decir que puede haber más de una clase padre de la cuál derive una clase. Pero son pocos los lenguajes de programación que utilizan herencia múltiple, como son C++ y Eiffel.

El concepto de visibilidad va unido al de herencia. Si bien, exceptuando constructores y destructores, todos los miembros son heredados, solo podrá accederse desde la clase hija a aquellos declarados como miembros públicos (+) o protegidos (#). Volveremos a este concepto más adelante.

Asociación, composición y agregación

A parte de las relaciones de herencia y generalización explicadas con anterioridad, también existen otra clase de relaciones y son una representación de una conexión semántica entre los elementos de un modelo.

La asociación es un tipo de relación en la que los objetos pueden pertenecer a la misma o a varias clases distintas, las cuales interactúan entre sí. Los objetos pueden existir independientemente de los otros, y se representa utilizando una línea continua.

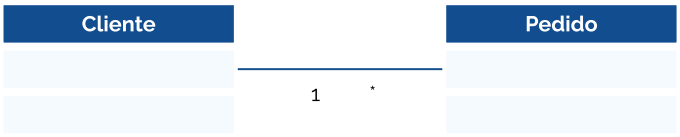

Imagen 5. Asociación.

Para representar relaciones entre un objeto "compuesto" y un objeto "componente", tenemos las siguientes relaciones:

- **Agregación:** este tipo de asociación es opcional, por lo que los componentes son capaces de existir por sí mismos y de participar de otras relaciones de composición con otros objetos. Estas relaciones también se conocen como composición débil y se representan con un rombo sin fondo.

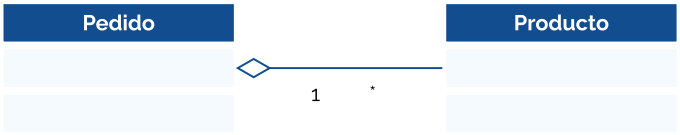

Imagen 6. Agregación.

- **Composición:** este tipo de asociación es obligatoria. Del compuesto dependen los componentes y sin él no tiene sentido que existan. Por tanto, el objeto compuesto debe destruir los componentes cuando este es destruido. Su representación consiste en un rombo con fondo negro.

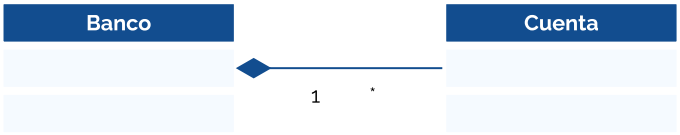

Imagen 7. Composición.

2.3.8. Polimorfismo

El polimorfismo es la propiedad que permite a los objetos adoptar distintas formas en tiempos de ejecución, además de permitir que se utilice como objeto de clase base un objeto de clase derivada.

El polimorfismo está estrechamente relacionado con la herencia, y ambos constituyen los pilares de la POO. Encontramos polimorfismo en las siguientes formas:

- **Asignación polimorfa:** sucede cuando se contiene en un elemento declarado de clase base un objeto de clase derivada. Para esto, el lenguaje debe permitir la relajación del sistema de tipos.
- **Ejecución polimorfa:** sucede al invocar métodos con el mismo nombre sobre instancias que son de diferente clase sin tener constancia de ello.

www.universae.com