3.A. Introducción y conceptos de la POO

Sitio: Formación Profesional a Distancia
Curso: Programación
Libro: 3.A. Introducción y conceptos de la POO
Imprimido por: Iván Jiménez Utiel
Día: jueves, 7 de noviembre de 2019, 15:17

Tabla de contenidos

1. Introducción.

Si nos paramos a observar el mundo que nos rodea, podemos apreciar que casi todo está formado por **objetos**. Existen coches, edificios, sillas, mesas, semáforos, ascensores e incluso personas o animales. **Todos ellos pueden ser considerados objetos, con una serie de características y comportamientos.** Por ejemplo, existen coches de diferentes marcas, colores, etc. y pueden acelerar, frenar, girar, etc., o las personas tenemos diferente color de pelo, ojos, altura y peso y podemos nacer, crecer, comer, dormir, etc.

Los programas son el resultado de la búsqueda y obtención de una solución para un problema del mundo real. Pero ¿en qué medida los programas están organizados de la misma manera que el problema que tratan de solucionar? La respuesta es que muchas veces los programas se ajustan más a los términos del sistema en el que se ejecutarán que a los del propio problema.

Si redactamos los programas utilizando los mismos términos de nuestro mundo real, es decir, utilizando objetos, y no los términos del sistema o computadora donde se vaya a ejecutar, conseguiremos que éstos sean más legibles y, por tanto, más fáciles de modificar.

Esto es precisamente lo que pretende la **Programación Orientada a Objetos (POO)**, en inglés **OOP (Object Oriented Programming),** establecer una serie de técnicas que permitan trasladar los problemas del mundo real a nuestro sistema informático. Ahora que ya conocemos la sintaxis básica de Java, es el momento de comenzar a utilizar las características orientadas a objetos de este lenguaje, y estudiar los conceptos fundamentales de este modelo de programación.

2. Fundamentos de la Programación Orientada a Objetos.

Dentro de las distintas formas de hacer las cosas en programación, distinguimos dos paradigmas fundamentales:

- **Programación Estructurada,** se crean **funciones y procedimientos** que definen las acciones a realizar, y que posteriormente forman los programas.

- **Programación Orientada a Objetos,** considera los programas en términos de **objetos** y todo gira alrededor de ellos.

Pero ¿en qué consisten realmente estos paradigmas? Veamos estos dos modelos de programación con más detenimiento. Inicialmente se programaba aplicando las técnicas de programación tradicional, también conocidas como **Programación Estructurada**. El problema se descomponía en unidades más pequeñas hasta llegar a acciones o verbos muy simples y fáciles de codificar. Por ejemplo, en la resolución de una ecuación de primer grado, lo que hacemos es descomponer el problema en acciones más pequeñas o pasos diferenciados:

- Pedir valor de los coeficientes.
- Calcular el valor de la incógnita.
- Mostrar el resultado.

Si nos damos cuenta, esta serie de acciones o pasos diferenciados no son otra cosa que verbos; por ejemplo el verbo pedir, calcular, mostrar, etc.

Sin embargo, la Programación Orientada a Objetos aplica de otra forma diferente la **técnica de programación "divide y vencerás"**. Este paradigma surge en un intento de salvar las dificultades que, de forma innata, posee el software. Para ello lo que hace es descomponer, en lugar de acciones, en objetos. El principal objetivo sigue siendo descomponer el problema en problemas más pequeños, que sean fáciles de manejar y mantener, fijándonos en cuál es el escenario del problema e intentando reflejarlo en nuestro programa. O sea, se trata de trasladar la visión del mundo real a nuestros programas. Por este motivo se dice que **la Programación Orientada a Objetos aborda los problemas de una forma más natural,** entendiendo como natural que está más en contacto con el mundo que nos rodea.

La Programación Estructurada se centra en el conjunto de acciones a realizar en un programa, haciendo una división de procesos y datos. La Programación Orientada a Objetos se centra en la relación que existe entre los datos y las acciones a realizar con ellos, y los encierra dentro del concepto de objeto, tratando de realizar una abstracción lo más cercana al mundo real.

La Programación Orientada a Objetos es un sistema o conjunto de reglas que nos ayudan a descomponer la aplicación en objetos. A menudo se trata de representar las entidades y objetos que nos encontramos en el mundo real mediante componentes de una aplicación. Es decir, debemos establecer una correspondencia directa entre el espacio del problema y el espacio de la solución. ¿Pero en la práctica esto qué quiere decir? Pues que a la hora de escribir un programa, nos fijaremos en los objetos involucrados, sus características comunes y las acciones que pueden realizar. Una vez localizados los objetos que intervienen en el problema real (espacio del problema), los tendremos que trasladar al programa informático (espacio de la solución). Con este planteamiento, la solución a un problema dado se convierte en una tarea sencilla y bien organizada.

Autoevaluación

Relaciona el término con su definición, escribiendo el número asociado a la definición en el hueco correspondiente.

Paradigma	Relación	Definición
Programación Orientada a Objetos.		1. Maneja funciones y procedimientos que definen las acciones a realizar.
Programación Estructurada.		2. Representa las entidades del mundo real mediante componentes de la aplicación.

Resolver

2.1. Conceptos.

Para entender mejor la filosofía de orientación a objetos veamos algunas características que la diferencian de las técnicas de programación tradicional.

En la Programación Estructurada, el programa estaba compuesto por un conjunto de **datos y funciones "globales".** El término global significaba que eran accesibles por todo el programa, pudiendo ser llamados en cualquier ubicación de la aplicación. Dentro de las funciones se situaban las instrucciones del programa que manipulaban los datos. **Funciones y datos se encontraban separados y totalmente independientes.** Esto ocasionaba dos problemas principales:

- Los programas se creaban y estructuraban de acuerdo con la arquitectura de la computadora donde se tenían que ejecutar.
- Al estar separados los datos de las funciones, éstos eran visibles en toda la aplicación. Ello ocasionaba que cualquier modificación en los datos podía requerir la modificación en todas las funciones del programa, en correspondencia con los cambios en los datos.

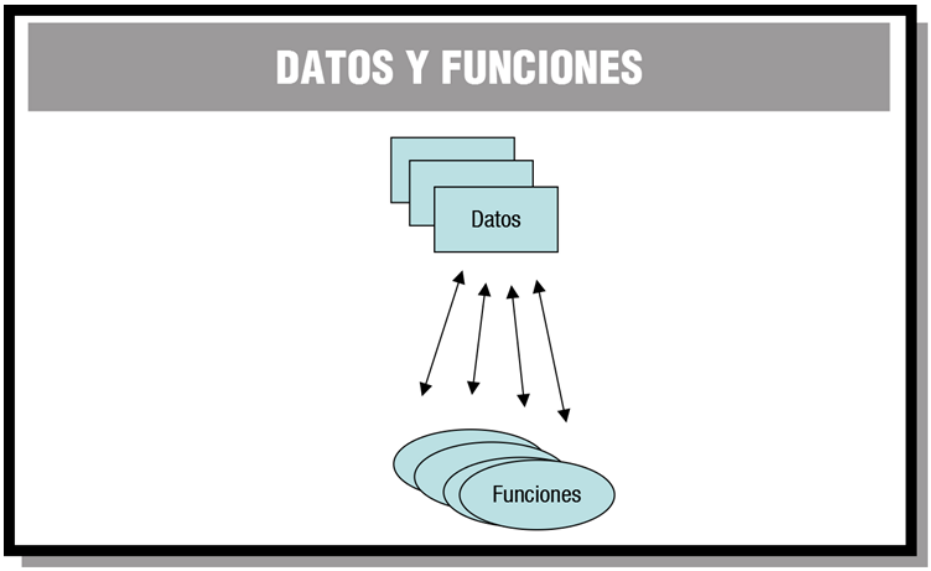

Imagen extraída de curso Programación del MECD.

En la **Programación Orientada a Objetos la situación es diferente**. La utilización de **objetos** permite un mayor nivel de **abstracción** que con la Programación Estructurada, y ofrece las siguientes diferencias con respecto a ésta:

- El programador organiza su programa en **objetos**, que son **representaciones del mundo real q**ue están más cercanas a la forma de pensar de la gente.
- Los datos, junto con las funciones que los manipulan, son parte interna de los objetos y no están accesibles al resto de los objetos. Por tanto, los cambios en los datos de un objeto sólo afectan a las funciones definidas para ese objeto, pero no al resto de la aplicación.

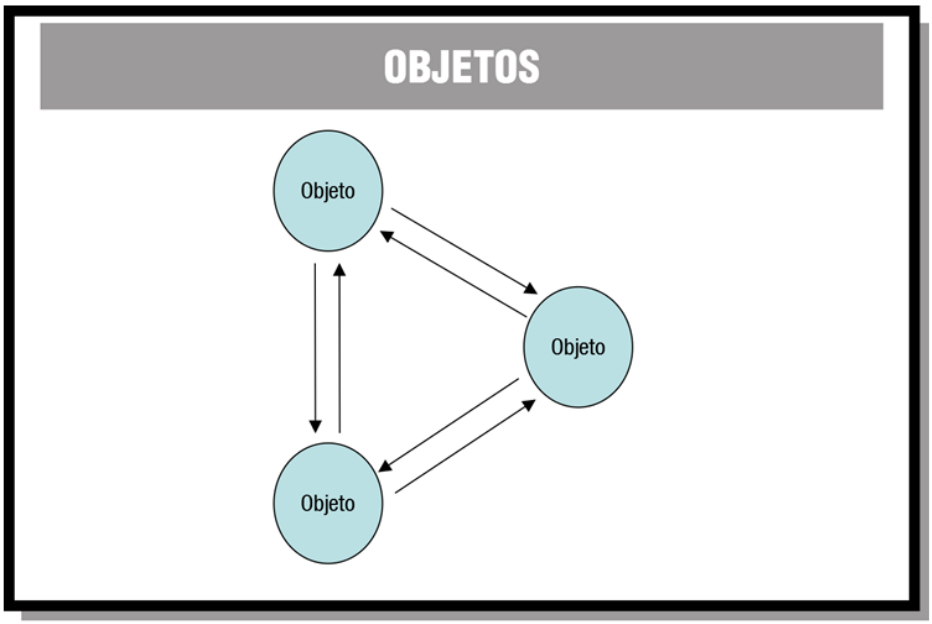

Imagen extraída de curso Programación del MECD.

Todos los programas escritos bajo el paradigma orientado a Objetos se pueden escribir igualmente mediante la Programación Estructurada. Sin embargo, la Programación Orientada a Objetos es la que mayor facilidad presenta para el desarrollo de programas basados en interfaces gráficas de usuario.

2.2. Beneficios.

Según lo que hemos visto hasta ahora, **un objeto es cualquier entidad que podemos ver o apreciar.** El concepto fundamental de la Programación Orientada a Objetos son, precisamente, los objetos. Pero ¿qué beneficios aporta la utilización de objetos? Fundamentalmente la posibilidad de representar el problema en términos del mundo real, que como hemos dicho están más cercanos a nuestra forma de pensar, pero existen otra serie de ventajas como las siguientes:

- **Comprensión**. Los conceptos del espacio del problema se hayan reflejados en el código del programa, por lo que la mera lectura del código nos describe la solución del problema en el mundo real.

- **Modularidad.** Facilita la modularidad del código, al estar las definiciones de objetos en módulos o archivos independientes, hace que las aplicaciones estén mejor organizadas y sean más fáciles de entender.

- **Fácil mantenimiento.** Cualquier modificación en las acciones queda automáticamente reflejada en los datos, ya que ambos están estrechamente relacionados. Esto hace que el mantenimiento de las aplicaciones, así como su corrección y modificación sea mucho más fácil. Por ejemplo, podemos querer utilizar un algoritmo más rápido, sin tener que cambiar el programa principal. Por otra parte, al estar las aplicaciones mejor organizadas, es más fácil localizar cualquier elemento que se quiera modificar y/o corregir. Esto es importante ya que se estima que los mayores costes de software no están en el proceso de desarrollo en sí, sino en el mantenimiento posterior de ese software a lo largo de su vida útil.

- **Seguridad**. La probabilidad de cometer errores se ve reducida, ya que no podemos modificar los datos de un objeto directamente, sino que debemos hacerlo mediante las acciones definidas para ese objeto. Imaginemos un objeto lavadora. Se compone de un motor, tambor, cables, tubos, etc. Para usar una lavadora no se nos ocurre abrirla y empezar a manipular esos elementos, ya que lo más probable es que se estropee. En lugar de eso utilizamos los programas de lavado establecidos. Pues algo parecido con los objetos, no podemos manipularlos internamente, sólo utilizar las acciones que para ellos hay definidas.

- **Reusabilidad**. Los objetos se definen como entidades reutilizables, es decir, que los programas que trabajan con las mismas estructuras de información, pueden reutilizar las definiciones de objetos empleadas en otros programas, e incluso las acciones definidas sobre ellos. Por ejemplo, podemos crear la definición de un objeto de tipo `persona` para una aplicación de negocios y deseamos construir a continuación otra aplicación, digamos de educación, en donde utilizamos también personas, no es necesario crear de nuevo el objeto, sino que por medio de la reusabilidad podemos utilizar el tipo de objeto `persona` previamente definido.

Imagen extraída de curso Programación del MECD.

2.3. Características.

Cuando hablamos de Programación Orientada a Objetos, existen una serie de características que se deben cumplir. Cualquier lenguaje de programación orientado a objetos las debe contemplar. Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción.** Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la **clase**. Básicamente, una clase es un tipo de dato que agrupa las características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global de todo el conjunto. Por ejemplo, si pensamos en una clase `Vehiculo` que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como `Coche` y `Camion`. Entonces se dice que `Vehiculo` es una abstracción de `Coche` y de `Camion`.

- **Modularidad.** Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.

- **Encapsulación.** También llamada **"ocultamiento de la información"**. La **encapsulación** o **encapsulamiento** es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto `Persona` y otro `Coche`**.** `Persona` se comunica con el objeto Coche para llegar a su destino, utilizando para ello las acciones que Coche tenga definidas como por ejemplo conducir. Es decir, `Persona` utiliza `Coche` pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.

- **Jerarquía**. Mediante esta propiedad podemos definir relaciones de jerarquías entre clases y objetos. Las dos jerarquías más importantes son la jerarquía **"es un"** llamada **generalización** o **especialización** y la jerarquía **"es parte de"**, llamada **agregación**. Conviene detallar algunos aspectos:
  - La generalización o especialización, también conocida como **herencia**, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase `CochedeCarreras` a partir de la clase `Coche`, y así sólo tendremos que definir las nuevas características que tenga.
  - La agregación, también conocida como **inclusión**, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un `Coche` está formado por `Motor`, `Ruedas`, `Frenos` y `Ventanas`. Se dice que `Coche` es una agregación y `Motor`, `Ruedas`, `Frenos` y `Ventanas` son agregados de `Coche`.

- **Polimorfismo**. Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase Animal y la acción de expresarse. Nos encontramos que cada tipo de `Animal` puede hacerlo de manera distinta, los `Perros` ladran, los `Gatos` maullan, las `Personas` hablamos, etc. Dicho de otra manera, el polimorfismo indica la posibilidad de tomar un objeto (de tipo `Animal`, por ejemplo), e indicarle que realice la acción de expresarse, esta acción será diferente según el tipo de mamífero del que se trate.

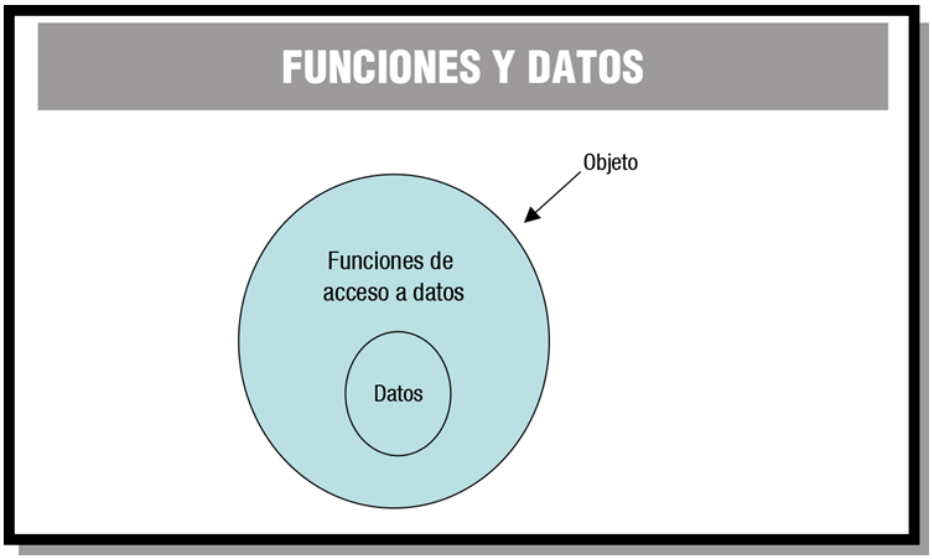

Imagen extraída de curso Programación del MECD.

2.4. Lenguajes de programación orientados a objetos.

Una panorámica de la evolución de los lenguajes de programación orientados a objetos hasta llegar a los utilizados actualmente es la siguiente:

- **Simula (1962)**. El primer lenguaje con objetos fue B1000 en 1961, seguido por Sketchpad en 1962, el cual contenía clones o copias de objetos. Sin embargo, fue Simula el primer lenguaje que introdujo el concepto de clase, como elemento que incorpora datos y las operaciones sobre esos datos. En 1967 surgió Simula 67 que incorporaba un mayor número de tipos de datos, además del apoyo a objetos.
- **SmallTalk (1972)**. Basado en Simula 67, la primera versión fue Smalltalk 72, a la que siguió Smalltalk 76, versión totalmente orientada a objetos. Se caracteriza por soportar las principales propiedades de la Programación Orientada a Objetos y por poseer un entorno que facilita el rápido desarrollo de aplicaciones. El Modelo-Vista-Controlador (MVC) fue una importante contribución de este lenguaje al mundo de la programación. El lenguaje Smalltalk ha influido sobre otros muchos lenguajes como C++ y Java.
- **C++ (1985).** C++ fue diseñado por Bjarne Stoustrup en los laboratorios donde trabajaba, entre 1982 y 1985. Lenguaje que deriva del C, al que añade una serie de mecanismos que le convierten en un lenguaje orientado a objetos. No tiene recolector de basura automática, lo que obliga a utilizar un destructor de objetos no utilizados. En este lenguaje es donde aparece el concepto de clase tal y como lo conocemos actualmente, como un conjunto de datos y funciones que los manipulan.
- **Eiffel (1986)**. Creado en 1985 por Bertrand Meyer, recibe su nombre en honor a la famosa torre de París. Tiene una sintaxis similar a C. Soporta todas las propiedades fundamentales de los objetos, utilizado sobre todo en ambientes universitarios y de investigación. Entre sus características destaca la posibilidad de traducción de código Eiffel a Lenguaje C. Aunque es un lenguaje bastante potente, no logró la aceptación de C++ y Java.
- **Java (1995)**. Diseñado por Gosling de Sun Microsystems a finales de 1995. Es un lenguaje orientado a objetos diseñado desde cero, que recibe muchas influencias de C++. Como sabemos, se caracteriza porque produce un bytecode que posteriormente es interpretado por la máquina virtual. La revolución de Internet ha influido mucho en el auge de Java.
- **C# (2000)**. El lenguaje C#, también es conocido como Sharp. Fue creado por Microsoft, como una ampliación de C con orientación a objetos. Está basado en C++ y en Java. Una de sus principales ventajas que evita muchos de los problemas de diseño de C++.

Autoevaluación

Relaciona los lenguajes de programación indicados con la característica correspondiente, escribiendo el número asociado a la característica en el hueco correspondiente.

Lenguaje de programación	Relación	Tiene la característica de que …
Java		1. Fue el primer lenguaje que introdujo el concepto de clase.
SmallTalk		2. Introdujo el concepto del Modelo-Vista-Controlador.
Simula		3. Produce un bytecode para ser interpretado por la máquina virtual.
C++		4. Introduce el concepto de clase tal cual lo conocemos, con atributos y métodos.

Resolver