ITEDES

Educación Digital

Módulo:

BackEnd

Segmento:

Programación Orientada a Objetos

Tema:

Introducción

Prof. Germán C. Basisty
german.basisty@itedes.com

Índice de Contenidos

Qué es la Programación Orientada a Objetos

La **programación orientada a objetos** (**POO**, u **OOP** según sus siglas en inglés) es un **paradigma de programación** que viene a innovar la forma de obtener resultados. Los **objetos** manipulan los **datos de entrada** para la obtención de **datos de salida** específicos, donde cada **objeto** ofrece una funcionalidad especial.

Muchos de los **objetos** pre-diseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

Está basada en varias técnicas, incluyendo **herencia**, **cohesión**, **abstracción**, **polimorfismo**, **acoplamiento** y **encapsulamiento**.

Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la **orientación a objetos**, como **Java**, **C#**, **C++** y **Python** entre otros.

Introducción

Los **objetos** son entidades que tienen un determinado **estado**, **comportamiento** e **identidad**:

La **identidad** es una propiedad de un **objeto** que lo diferencia del resto; dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

Un **objeto** contiene toda la información que permite definirlo e identificarlo frente a otros **objetos** pertenecientes a otras **clases** e incluso frente a **objetos** de una misma **clase**, al poder tener valores bien diferenciados en sus **atributos**. A su vez, los **objetos** disponen de mecanismos de interacción llamados **métodos**, que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de **estado** en los propios **objetos**. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el **estado** y el **comportamiento**.

Los **métodos** (**comportamiento**) y **atributos** (**estado**) están estrechamente relacionados por la **propiedad de conjunto**. Esta propiedad destaca que una **clase** requiere de **métodos** para poder tratar los **atributos** con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. *Hacerlo podría producir el hábito erróneo de crear **clases** contenedoras de información por un lado y **clases** con métodos que manejen a las primeras por el otro*.

La **programación orientada a objetos** difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada sólo se escriben funciones que procesan datos. Los programadores que emplean **POO**, en cambio, primero definen **objetos** para luego enviarles **mensajes**.

Origen

Los conceptos de la **POO** tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard, del Centro de Cómputo Noruego en Oslo. En este centro se trabajaba en simulaciones de naves, que fueron confundidas por la explosión combinatoria de cómo las diversas cualidades de diferentes naves podían afectar unas a las otras. La idea surgió al **agrupar** los diversos tipos de naves en diversas **clases de objetos**, siendo responsable cada **clase de objetos** de definir sus propios datos y comportamientos. Fueron refinados más tarde en Smalltalk, desarrollado en Simula pero diseñado para ser un sistema completamente dinámico en el cual los **objetos** se podrían crear y modificar sobre la marcha (en tiempo de ejecución) en lugar de tener un sistema basado en programas estáticos.

La **POO** se fue convirtiendo en el estilo de programación dominante a mediados de los años 1980, en gran parte debido a la influencia de **C++**, una extensión del lenguaje de programación **C**. Su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para las cuales la **POO** está particularmente bien adaptada. En este caso, se habla también de programación dirigida por eventos.

Las **características de orientación a objetos** fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo **Ada**, **BASIC**, **Lisp** y **Pascal** entre otros. La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y en la capacidad de mantenimiento del código. Los **lenguajes orientados a objetos** puros, por su parte, carecían de las características de las cuales muchos programadores habían venido a depender. Para saltar este obstáculo, se hicieron muchas tentativas para crear **nuevos lenguajes basados en métodos orientados a objetos**, pero permitiendo algunas características imperativas de maneras seguras. El lenguaje de programación Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos, pero ahora ha sido esencialmente reemplazado por **Javamáquina virtual Java** en la mayoría de las plataformas.

Conceptos fundamentales

La **POO** es una forma de programar que trata de encontrar una solución a los problemas de la programación estructurada. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

Clase

Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.

Herencia

Por ejemplo, herencia de la clase C a la clase D, es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables registrados como públicos (public) en C. Los componentes registrados como privados (private) también se heredan pero se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Para poder acceder a un atributo u operación de una clase en cualquiera de sus subclases pero mantenerla oculta para otras clases es necesario registrar los componentes como protegidos (protected), de esta manera serán visibles en C y en D pero no en otras clases.

Objeto

Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema.

Método

Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

Evento

Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

Atributos

Características que tiene la clase.

Mensaje

Es la petición de un objeto a otro para solicitar la ejecución de alguno de sus métodos o para obtener el valor de un atributo.

Estado interno

El estado de un objeto está determinado por los valores que poseen sus atributos en un momento dado.

Miembros de clase / objeto

Atributos, identidad, relaciones y métodos.

Características de la POO

Existe un acuerdo acerca de qué características contempla la orientación a objetos:

Abstracción

Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar "cómo" se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos, y, cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

Encapsulamiento

Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión (diseño estructurado) de los componentes del sistema.

Polimorfismo

Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O, dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en tiempo de ejecución, esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en tiempo de compilación) de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Herencia

Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple; siendo de alta complejidad técnica por lo cual suele recurrirse a la herencia virtual para evitar la duplicación de datos.

Modularidad

Se denomina "modularidad" a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la modularidad de diversas formas.

Principio de ocultación

Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interface a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas.

Recolección de basura

La recolección de basura (garbage collection) es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos.

Corrientes de la POO

Para realizar programación orientada a objetos existen dos corrientes principales:

Basarse en clases

Es la más ampliamente usada por los lenguajes de programación orientada a objetos POO. Por ejemplo es usada por **Java**, **C++** y **C#**. Se basa en crear una estructura molde llamada clase donde se especifican los campos y métodos que tendrán nuestros objetos. Cada vez que necesitamos un objeto creamos una instancia (o copia del objeto) usando la clase como molde.

Basarse en prototipos

Es soportado en **Javascript**, **Python** y **Ruby** entre otros lenguajes. No hay clases, solo hay objetos. El mecanismo para la reutilización está dado por la clonación de objetos. Se crean directamente objetos y cuando se quiere generar otro con la misma estructura se usa clonación. Una vez clonado si queremos podemos agregar los campos y métodos necesarios. Un objeto prototípico es un objeto que se utiliza como una plantilla a partir de la cual se obtiene el conjunto inicial de propiedades de un objeto. Cualquier objeto puede ser utilizado como el prototipo de otro objeto, permitiendo al segundo objeto compartir las propiedades del primero.

Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como **WORA**, o write once, run anywhere), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser compilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados.

El lenguaje de programación Java fue originalmente desarrollado por James Gosling, de Sun Microsystems (la cual fue adquirida por la compañía Oracle), y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Filosofía

El lenguaje **Java** se creó con cinco objetivos principales:

- Debería usar el paradigma de la programación orientada a objetos.
- Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
- Debería incluir por defecto soporte para trabajo en red.
- Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
- Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como **C++**.

Sintaxis

La sintaxis de **Java** se deriva en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, **Java** fue construido desde el principio para ser completamente orientado a objetos. Todo en **Java** es un objeto (salvo algunas excepciones), y todo en **Java** reside en alguna clase (recordemos que una clase es un molde a partir del cual pueden crearse varios objetos).

Ejemplo

```
public class Dog {
    private String name;
    private Integer age;
    private Double weight;

    public Dog(String name, Double weight) {
        this.name = name;
        this.weight = weight;
        age = 0;
    }

    public void setName(String name) {
        if(name.length() <= 0)
            System.out.println("Nombre no valido");
        else
            this.name = name;
    }

    public String getName() {
        return name;
    }

    public Double getWeight() {
        return weight;
    }

    public void eat(Double food) {
        weight += food;
    }

    public void poop(Double shit) {
        weight -= shit;
    }

    public Integer getAge() {
        return age;
    }

    public void getOlder() {
        age++;
    }

    public static void bark() {
        System.out.println("guau guau!");
    }
}
```

C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por **Microsoft** como parte de su plataforma **.NET**, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). **C#** es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Aunque **C#** forma parte de la plataforma **.NET**, ésta es una **API**, mientras que **C#** es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco **.Net Core**, el cual genera programas para distintas plataformas como **Microsoft Windows**, **Android**, **iOS**, **Mac OS** y **GNU/Linux**.

Filosofía

- Lenguaje de programación orientado a objetos simple, moderno y de propósito general.
- Inclusión de principios de ingeniería de software tales como revisión estricta de los tipos de datos, revisión de límites de vectores, detección de intentos de usar variables no inicializadas, y recolección de basura automática.
- Capacidad para desarrollar componentes de software que se puedan usar en ambientes distribuidos.
- Portabilidad del código fuente.
- Fácil migración del programador al nuevo lenguaje, especialmente para programadores familiarizados con C, C++ y Java.
- Soporte para internacionalización.
- Adecuación para escribir aplicaciones de cualquier tamaño: desde las más grandes y sofisticadas como sistemas operativos hasta las más pequeñas funciones.
- Aplicaciones económicas en cuanto a memoria y procesado.

Sintaxis

Su sintaxis básica deriva de **C/C++** y utiliza el modelo de objetos de la plataforma **.NET**, similar al de **Java**, aunque incluye mejoras derivadas de otros lenguajes.

Ejemplo

```
using System;

public class Coche
{
    private int numPuertas;
    public int NumPuertas
    {
        get
        {
            return this.numPuertas;
        }
        set
        {
            this.numPuertas = value;
        }
    }

    public Coche(int numPuertas)
    {
        this.NumPuertas = numPuertas;
    }

    public Coche() : this(2)
    {
    }
}
```

Python

Python es un lenguaje de programación cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Filosofía

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.19
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.

Sintaxis

Python fue diseñado para ser leído con facilidad. Una de sus características es el uso de palabras donde otros lenguajes utilizarían símbolos. Por ejemplo, los operadores lógicos !, || y && en Python se escriben not, or y and, respectivamente.

El contenido de los bloques de código es delimitado mediante espacios o tabuladores, conocidos como indentación, antes de cada línea de órdenes pertenecientes al bloque. **Python** se diferencia así de otros lenguajes de programación que mantienen como costumbre declarar los bloques mediante un conjunto de caracteres, normalmente entre llaves. Se pueden utilizar tanto espacios como tabuladores para indentar el código, pero se recomienda no mezclarlos.

Ejemplo

```
class Alumno:
    'Clase para alumnos'
    numalumnos = 0
    sumanotas = 0

    def __init__(self, nombre, nota):
        self.nombre = nombre
        self.nota = nota
        Alumno.numalumnos += 1
        Alumno.sumanotas += nota

    def mostrarNombreNota(self):
        return(self.nombre, self.nota)

    def mostrarNumAlumnos(self):
        return(Alumno.numalumnos)

    def mostrarSumaNotas(self):
        return(Alumno.sumanotas)

    def mostrarNotaMedia(self):
        if Alumno.numalumnos > 0:
            return(Alumno.sumanotas/Alumno.numalumnos)
        else:
            return("Sin alumnos")
```