Programación orientada a objetos

Autor:

Kevin Cárdenas.

2023

Índice

1. Introducción

La programación orientada a objetos ("POO") es un paradigma de programación que se enfoca en la creación de objetos que contienen tanto datos como funciones que actúan sobre esos datos. Esta técnica de programación se centra en la creación de *clases* que definen los objetos, y los objetos son instancias de esas clases.

En POO, los objetos son considerados como "cosas" que tienen ciertas propiedades y comportamientos. Cada objeto puede tener un conjunto de datos asociados (denominados "atributos" o "propiedades"), así como una serie de funciones o "métodos" que pueden realizar acciones sobre esos datos.

Una de las principales ventajas de la POO es que permite la modularidad del código, lo que significa que puedes dividir el código en pequeñas secciones que se pueden desarrollar, probar y mantener de forma independiente. Además, la POO también hace que el código sea más fácil de entender y mantener a largo plazo, ya que las funciones y datos relacionados están agrupados en objetos que tienen un propósito claro.

1.1. Encapsulación, Herencia, Polimorfismo y Abstracción

En POO, hay cuatro conceptos fundamentales que se deben entender:

1. *Encapsulación*: este concepto se refiere a la ocultación de datos y métodos dentro de un objeto para que solo puedan ser accedidos por el objeto mismo o por otros objetos que tengan una relación específica con él.

2. *Herencia*: la herencia permite crear una clase nueva a partir de una ya existente, pero añadiéndole o modificando sus propiedades y métodos.

3. *Polimorfismo*: este concepto se refiere a la capacidad de diferentes objetos de una misma clase para responder de manera distinta a un mismo mensaje o función.

4. *Abstracción*: la abstracción se refiere a la capacidad de crear una clase que representa un concepto o idea abstracta, sin necesidad de especificar todos los detalles en su definición.

La POO se utiliza en muchos lenguajes de programación populares, como Java, Python, C++ y muchos otros

JAVA orientado a objetos

Java es uno de los lenguajes de programación más populares y utilizados en el mundo, y también es conocido por su fuerte orientación a objetos. En Java, todo es un objeto, incluso los tipos primitivos, como

los enteros y los caracteres, se representan como objetos.

La POO en Java se basa en la definición de clases y objetos. Una *clase* es un plano o plantilla que define la estructura, atributos y métodos de los objetos que se pueden crear a partir de ella. Un *objeto* es una instancia de una clase, es decir, un miembro concreto de la clase que contiene sus propios valores para los atributos definidos en la clase.

La encapsulación en Java se logra mediante el uso de modificadores de acceso, como `private`, `protected` y `public`, que permiten controlar el acceso a los atributos y métodos de una clase desde otras clases. La herencia se logra mediante la extensión de una clase existente mediante la palabra clave `extends`. El polimorfismo se logra mediante la implementación de interfaces o mediante la sobrecarga y la anulación de métodos.

La abstracción en Java se logra mediante el uso de interfaces y clases abstractas. Una *interface* es una colección de métodos sin implementación que una clase puede implementar para proporcionar su propia implementación. Una *clase abstracta* es una clase que no se puede instanciar directamente, sino que debe ser subclasificada y extendida.

Python orientado a objetos

Python es un lenguaje de programación de alto nivel y fácil de aprender que también es conocido por su fuerte orientación a objetos. En Python, todo es un objeto, desde los números y las cadenas hasta las funciones y las clases.

La POO en Python se basa en la definición de clases y objetos. Una *clase* es un plano o plantilla que define la estructura, atributos y métodos de los objetos que se pueden crear a partir de ella. Un *objeto* es una instancia de una clase, es decir, un miembro concreto de la clase que contiene sus propios valores para los atributos definidos en la clase.

La encapsulación en Python se logra mediante el uso de convenciones de nomenclatura, como el uso de guiones bajos (`_`) o doble guiones bajos (`__`) para indicar la visibilidad de los atributos y métodos. La herencia se logra mediante la herencia de una clase existente mediante la palabra clave `class ClassName(ParentClass)`. El polimorfismo se logra mediante la implementación de métodos con el mismo nombre en diferentes clases.

La abstracción en Python se logra mediante el uso de clases abstractas y métodos abstractos. Una *clase abstracta* es una clase que no se puede instanciar directamente, sino que debe ser subclasificada y extendida. Un *método abstracto* es un método que se define en una clase abstracta pero no se implementa, lo que significa que las subclases deben proporcionar su propia implementación.

Python también es conocido por su soporte para la programación orientada a objetos basada en funciones. En este enfoque, las funciones se utilizan para encapsular la lógica de la clase y para proporcionar una interfaz más clara para el uso de la clase.

1.2. Modelado

El modelado es una parte fundamental de la programación orientada a objetos. El objetivo del modelado es crear un modelo conceptual de un sistema o aplicación en términos de clases, objetos, atributos, métodos y relaciones entre ellos.

En el modelado, se utilizan diagramas de clases para representar la estructura de un sistema. Un diagrama de clases es una representación gráfica de las clases en un sistema y las relaciones entre ellas. Las clases se representan como cajas con el nombre de la clase, sus atributos y métodos. Las relaciones entre las clases se representan mediante líneas que indican la naturaleza de la relación.

Además de los diagramas de clases, también se utilizan otros diagramas para modelar otros aspectos de la programación orientada a objetos. Por ejemplo, el diagrama de secuencia se utiliza para modelar la interacción entre objetos en un sistema, el diagrama de estado se utiliza para modelar el comportamiento de un objeto en diferentes estados y el diagrama de actividades se utiliza para modelar el flujo de trabajo de un proceso.

El modelado es una parte importante del desarrollo de software orientado a objetos porque ayuda a los desarrolladores a comprender mejor el sistema que están construyendo y a identificar y resolver problemas potenciales antes de escribir cualquier código. También ayuda a garantizar que el sistema sea modular, fácil de entender y mantener a largo plazo.

UML - Lenguaje de Modelado Unificado

UML ("Lenguaje de Modelado Unificado") es un lenguaje de modelado visual utilizado para el diseño y la documentación de sistemas de software orientados a objetos. UML es una herramienta popular utilizada por los desarrolladores de software para visualizar, especificar, construir y documentar los sistemas de software.

El lenguaje UML consta de varios tipos de diagramas, cada uno de los cuales se utiliza para modelar diferentes aspectos de un sistema de software. Estos diagramas incluyen:

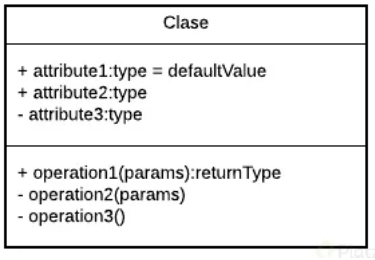
1. Diagrama de Clases: representa la estructura estática del sistema, incluyendo las clases, los atributos y los métodos, y las relaciones entre ellos.

2. Diagrama de Objetos: representa una instancia específica de una clase en un punto específico del tiempo, incluyendo los valores de los atributos.
3. Diagrama de Casos de Uso: representa los actores, los casos de uso y las relaciones entre ellos.
4. Diagrama de Secuencia: representa la interacción entre los objetos en una secuencia de eventos.
5. Diagrama de Actividad: representa el flujo de trabajo de un proceso o algoritmo.
6. Diagrama de Estados: representa el comportamiento de un objeto en diferentes estados.
7. Diagrama de Componentes: representa los componentes del sistema y las relaciones entre ellos.
8. Diagrama de Despliegue: representa la distribución física del sistema en el hardware.

Estos diagramas son útiles para modelar diferentes aspectos de un sistema de software y proporcionan una visión completa del sistema desde diferentes perspectivas. Los diagramas de UML son útiles tanto para los desarrolladores como para los interesados en el sistema, como los gerentes de proyecto y los clientes, ya que proporcionan una visión clara y detallada del sistema.

Clases

Las clases se representan así:



En la parte superior se colocan los atributos o propiedades, y debajo las operaciones de la clase. Notarás que el primer caracter con el que empiezan es un símbolo. Este denotará la visibilidad del atributo o método, esto es un término que tiene que ver con Encapsulamiento y veremos más adelante a detalle.
Estos son los niveles de visibilidad que puedes tener:

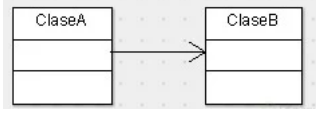
1. − private
2. + public
3. # protected
4. default

Una forma de representar las relaciones que tendrá un elemento con otro es a través de las flechas en UML, y aquí tenemos varios tipos, estos son los más comunes:

Asociación

Como su nombre lo dice, notarás que cada vez que esté referenciada este tipo de flecha significará que ese elemento contiene al otro en su definición.

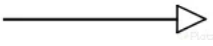
La flecha apuntará hacia la dependencia.



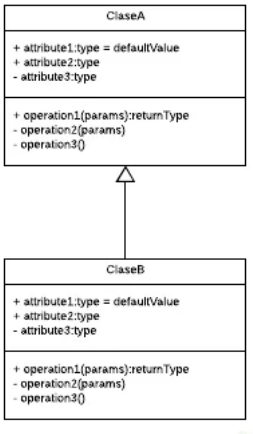
Con esto vemos que la ClaseA está asociada y depende de la ClaseB.

Herencia

Siempre que veamos este tipo de flecha se estará expresando la herencia.



La dirección de la flecha irá desde el hijo hasta el padre.



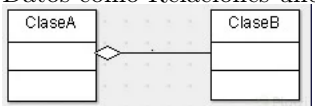
Con esto vemos que la ClaseB hereda de la ClaseA

Agregación

Este se parece a la asociación en que un elemento dependerá del otro, pero en este caso será: Un elemento dependerá de muchos otros.

Aquí tomamos como referencia la multiplicidad del elemento. Lo que comúnmente conocerías en Bases de

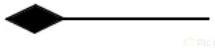
Datos como Relaciones uno a muchos.



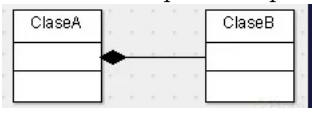
Con esto decimos que la ClaseA contiene varios elementos de la ClaseB. Estos últimos son comúnmente representados con listas o colecciones de datos.

Composición

Este es similar al anterior solo que su relación es totalmente compenetrada.



De tal modo que conceptualmente una de estas clases no podría vivir si no existiera la otra.



1.3. Buenas practicas

Aquí hay algunas buenas prácticas a seguir al trabajar con programación orientada a objetos:

1. Utilizar la encapsulación: La encapsulación es una técnica que ayuda a ocultar los detalles de implementación de una clase y permite que los objetos interactúen entre sí solo a través de interfaces definidas. Esto ayuda a reducir el acoplamiento y hace que el código sea más fácil de mantener.

2. Diseñar clases cohesivas: Una clase cohesiva es aquella que tiene una única responsabilidad claramente definida y no hace más de lo que se espera de ella. Al diseñar clases cohesivas, se reduce la complejidad del código y se facilita la comprensión y el mantenimiento.

3. Utilizar la herencia con precaución: La herencia es una técnica útil para compartir código común entre clases relacionadas. Sin embargo, se debe tener cuidado al usarla para evitar la creación de clases altamente acopladas y difíciles de mantener.

4. Evitar la duplicación de código: La duplicación de código puede llevar a errores y dificultar el mantenimiento. Por lo tanto, se deben buscar formas de reutilizar el código existente y evitar la duplicación innecesaria

5. Mantener el código limpio y legible: Un código limpio y legible es más fácil de entender, mantener y escalar. Se deben seguir buenas prácticas de codificación, como nombrar variables y métodos de manera

descriptiva, utilizar comentarios y espacios en blanco adecuados, y seguir un estilo de codificación consistente en todo el proyecto.

6. Seguir los principios SOLID: Los principios SOLID son un conjunto de reglas que ayudan a garantizar que el código sea fácil de entender, mantener y escalar. Estos principios incluyen la responsabilidad única, la abstracción, la inversión de dependencias, la sustitución de Liskov y la segregación de interfaces.

   los principios SOLID: Los principios SOLID son un conjunto de reglas que ayudan a garantizar que el código sea fácil de entender, mantener y escalar. Estos principios incluyen la responsabilidad única, la abstracción, la inversión de dependencias, la sustitución de Liskov y la segregación de interfaces.

   - Principio de responsabilidad única (SRP): Cada clase o módulo debe tener una sola responsabilidad. Esto significa que la clase o módulo debe tener una sola razón para cambiar. Si una clase o módulo tiene más de una responsabilidad, es más difícil de entender y mantener.
   - Principio de abierto/cerrado (OCP): Las clases o módulos deben estar abiertos para su extensión, pero cerrados para su modificación. Esto significa que si necesitamos agregar una nueva funcionalidad, debemos hacerlo a través de la extensión del código existente, en lugar de modificarlo directamente.
   - Principio de sustitución de Liskov (LSP): Las clases derivadas deben ser sustituibles por sus clases base. Esto significa que una instancia de una clase derivada debe ser capaz de ser usada en cualquier lugar donde se espera una instancia de la clase base, sin que se produzcan errores o comportamientos inesperados.
   - Principio de segregación de la interfaz (ISP): Los clientes no deben ser obligados a depender de interfaces que no usan. Esto significa que una clase o módulo no debería tener que depender de métodos o interfaces que no necesita.
   - Principio de inversión de dependencia (DIP): Las clases de alto nivel no deben depender de las clases de bajo nivel. Ambas clases deben depender de abstracciones. Esto significa que el código de alto nivel no debería depender directamente del código de bajo nivel, sino de una interfaz o abstracción que encapsule el comportamiento del código de bajo nivel.

1.4. Metodo constructor

En programación orientada a objetos, un constructor es un método especial utilizado para inicializar un objeto recién creado una vez que se ha reservado memoria para él. El constructor se llama automática-

mente al crear un objeto y es responsable de asignar valores iniciales a los atributos de la clase. En Java, el constructor se define con el nombre de la clase y no tiene tipo de retorno.

A continuación, se muestra un ejemplo de un constructor en Java:

```java
public class Persona {
    private String nombre;
    private int edad;
    private String direccion;

    public Persona(String nombre, int edad, String direccion) {
        this.nombre = nombre;
        this.edad = edad;
        this.direccion = direccion;
    }

    // metodos getters y setters

    public static void main(String[] args) {
        Persona persona = new Persona("Juan", 30, "Calle 123, Ciudad");

        System.out.println("Nombre: " + persona.getNombre());
        System.out.println("Edad: " + persona.getEdad());
        System.out.println("Direccion: " + persona.getDireccion());

        persona.setEdad(31);
        System.out.println("Nueva edad: " + persona.getEdad());
    }
}
```

En este ejemplo, la clase Persona tiene tres atributos: nombre, edad y direccion. Además, tiene un constructor que recibe estos tres parámetros y los asigna a los atributos correspondientes.

En el método main, se crea una instancia de la clase Persona utilizando el constructor que recibe los valores "Juan", 30 y Çalle 123, Ciudad". Luego, se imprimen los valores de los atributos utilizando los métodos getters correspondientes. Finalmente, se modifica la edad utilizando el método setter correspondiente y se vuelve a imprimir el valor de la edad para verificar que se ha modificado correctamente.

Te presento el ejemplo de un método constructor en Python:

```python
class Persona:
    def __init__(self, nombre, edad, direccion):
        self.nombre = nombre
```

```
        self.edad = edad
        self.direccion = direccion

    def get_nombre(self):
        return self.nombre

    def set_nombre(self, nombre):
        self.nombre = nombre

    def get_edad(self):
        return self.edad

    def set_edad(self, edad):
        self.edad = edad

    def get_direccion(self):
        return self.direccion

    def set_direccion(self, direccion):
        self.direccion = direccion


# Creamos una instancia de la clase Persona
persona = Persona("Juan", 30, "Calle 123, Ciudad")

# Accedemos a los atributos y metodos de la instancia
print("Nombre: ", persona.get_nombre())
print("Edad: ", persona.get_edad())
print("Direccion: ", persona.get_direccion())

# Modificamos la edad de la persona
persona.set_edad(31)

# Volvemos a imprimir los atributos de la instancia
print("Nueva edad: ", persona.get_edad())
```

En este ejemplo, creamos la clase "Persona" con un método constructor `__init__` que inicializa los atributos de la clase con los valores proporcionados al crear una instancia de la clase. También definimos los métodos get y set para acceder y modificar los atributos de la clase.

Luego, creamos una instancia de la clase "Persona" con el nombre "Juan", la edad 30 y la dirección

“Calle 123, Ciudad”. Accedemos a los atributos y métodos de la instancia para imprimir su información y modificar su edad.

2. JAVA orientada a objetos

Java es un lenguaje de programación orientado a objetos (POO) que se ha convertido en uno de los más populares en la actualidad. Su éxito se debe en gran medida a su capacidad para manejar de manera eficiente la programación orientada a objetos.

2.1. Clase

Definamos una clase, el formato es el siguiente:

```
public class Person {
    String name = ""
    void walk(){

    }
}
```

Este fregmento define una clase llamada “Person” que tiene un atributo “name” de tipo String y un método “walk” que no hace nada.

Por ejemplo:

```
public class Persona {
    private String nombre;
    private int edad;
    private String direccion;

    public Persona(String nombre, int edad, String direccion) {
        this.nombre = nombre;
        this.edad = edad;
        this.direccion = direccion;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
```

```
22      }
23
24      public void setEdad(int edad) {
25          this.edad = edad;
26      }
27
28      public String getDireccion() {
29          return direccion;
30      }
31
32      public void setDireccion(String direccion) {
33          this.direccion = direccion;
34      }
35  }
```

Este es un ejemplo de una clase “Persona” que tiene tres atributos privados (nombre, edad y dirección) y los métodos públicos necesarios para acceder y modificar estos atributos.

En la línea 1, se define la clase pública “Persona”. Los atributos “nombre”, “edad” y “direccion” son definidos en las líneas 2-4 como variables de tipo privado para protegerlos de accesos no deseados.

A continuación, se define el constructor de la clase en las líneas 6-10. Este constructor tiene tres parámetros (nombre, edad y dirección) y se utiliza para crear una nueva instancia de la clase “Persona”. Dentro del constructor, se inicializan los atributos de la clase con los valores recibidos en los parámetros utilizando la palabra clave “this”.

Luego, se definen los métodos “get” y “set” para acceder y modificar los valores de los atributos de la clase en las líneas 12-29. Cada método “get” devuelve el valor correspondiente del atributo, mientras que cada método “set” actualiza el valor del atributo con el valor proporcionado como parámetro.

Por último, se define el método “main” en las líneas 31-38. Este es el punto de entrada del programa y se utiliza para crear una instancia de la clase “Persona” con valores iniciales y para imprimir los valores de los atributos utilizando los métodos “get”. También se utiliza el método “setEdad” para actualizar la edad de la persona y se vuelve a imprimir para confirmar que el valor ha sido actualizado correctamente.

3. Python orientado a objetos

3.1. Clase

```
class Person:
    String name = ""
    void walk():
```

Este fregmento define una clase llamada "Person"que tiene un atributo "name"de tipo String y un método "walk"que no hace nada.

4. Bibliografía

Referencias

[1] Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[2] Platzi. *Curso de Java Básico*. Platzi, 2023.

https://platzi.com/cursos/java-basico/

[3] Platzi. *Curso de Java Orientado a Objetos*. Platzi, 2023.

https://platzi.com/cursos/java-poo/

[4] Oracle Corporation. *Java SE Documentation*. Oracle Corporation, 2023.

https://docs.oracle.com/en/java/javase/index.html