## Conceptos fundamentales de orientación a objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en la representación de objetos y sus interacciones para resolver problemas de software. La POO ha revolucionado la forma en que se aborda y resuelve los problemas en el desarrollo de software, ofreciendo una mayor modularidad, reutilización, abstracción y flexibilidad en el diseño y desarrollo de sistemas informáticos.

Los conceptos fundamentales de la POO son la abstracción, el encapsulamiento, la herencia y el polimorfismo. Estos conceptos son la base sobre la que se construyen las soluciones en POO, y su comprensión es crucial para cualquier programador que quiera utilizar este paradigma.

### *Abstracción*

La abstracción es un concepto fundamental de la POO que se refiere a la capacidad de simplificar un problema al ocultar detalles innecesarios y enfocarse en los aspectos relevantes. En el contexto de nuestra aplicación de gestión de mascotas, podemos utilizar la abstracción para representar a cada mascota como un objeto independiente, con sus propias propiedades y comportamientos.

Por ejemplo, podemos definir una clase "Mascota" que tenga propiedades como "nombre", "edad", "especie", "raza" y "dueño". También podemos definir métodos como "alimentar", "jugar" y "bañar" que representen los comportamientos que una mascota puede realizar. Al definir una clase "Mascota" de esta manera, estamos abstrayendo el concepto de una mascota a un nivel más alto, lo que nos permite trabajar con él de manera más eficiente y modular.

Además, la abstracción también nos permite modelar las relaciones entre los objetos en nuestra aplicación. Por ejemplo, podemos definir una clase "Dueño" que tenga propiedades como "nombre", "correo electrónico" y "dirección", y luego relacionarla con la clase "Mascota" a través de una propiedad "dueño". De esta manera, podemos abstraer la relación entre una mascota y su dueño en un objeto independiente y reutilizable.

### *Encapsulamiento*

Otro concepto fundamental en la Programación Orientada a Objetos es el encapsulamiento. El encapsulamiento es el mecanismo que nos permite ocultar los detalles internos de un objeto y exponer solo la interfaz pública para interactuar con él. Esto significa que podemos controlar el acceso a los datos y comportamientos de un objeto, asegurando que solo se puedan manipular de la manera prevista.

Para entender mejor este concepto, podemos imaginar un objeto "Mascota" que tenga un atributo "edad". Si queremos que la edad de la mascota solo se pueda modificar mediante un método específico, podemos hacer que el atributo sea privado y crear un método "setEdad" que se encargue de modificarlo. De esta manera, podemos asegurar que la edad de la mascota solo se modifique de la manera que deseamos y evitamos errores o cambios no deseados.

El encapsulamiento no solo permite un mayor control sobre el comportamiento de los objetos, sino que también facilita la modificación y mantenimiento del código. Al ocultar los detalles internos de un objeto, podemos cambiar su implementación sin afectar el resto del código que lo utiliza. Esto significa que podemos realizar cambios en el comportamiento de un objeto sin tener que modificar todo el código que lo utiliza, lo que mejora la modularidad y la reutilización del código.

Por ejemplo, si queremos agregar un nuevo atributo "color" a nuestra clase "Mascota", podemos hacerlo sin afectar a las partes del código que ya utilizan la clase. Solo tendríamos que modificar la implementación de la clase "Mascota" y asegurarnos de que los métodos que exponen la interfaz pública de la clase sigan funcionando correctamente. De esta manera, podemos mejorar la funcionalidad de nuestro código sin tener que preocuparnos por romper otras partes del programa.

### *Herencia*

En el contexto de nuestro sistema de gestión de mascotas, la herencia es un concepto clave de la Programación Orientada a Objetos que nos permite modelar relaciones jerárquicas entre objetos y promueve la reutilización de código.

La herencia es un mecanismo mediante el cual una clase puede heredar atributos y comportamientos de otra clase. La clase que hereda se llama subclase o clase derivada, mientras que la clase que proporciona los atributos y comportamientos se llama clase base o clase padre.

En nuestro sistema de gestión de mascotas, podríamos tener una clase base llamada "Mascota", que contenga atributos como "nombre", "edad" y "especie", y comportamientos como "alimentar" y "jugar". Luego, podríamos crear subclases específicas para cada tipo de mascota, como "Perro", "Gato" y "Pez", que hereden los atributos y comportamientos de la clase base "Mascota". Cada subclase también podría tener sus propios atributos y comportamientos únicos, como "ladra" para la clase "Perro" o "maulla" para la clase "Gato".

La herencia permite una mayor organización y reutilización de código al evitar la duplicación de atributos y comportamientos comunes en varias clases. También nos permite modelar relaciones jerárquicas entre objetos y simplificar la comprensión y el mantenimiento del código. Por ejemplo, si necesitamos agregar un nuevo comportamiento a todas las mascotas, simplemente lo agregamos a la clase base "Mascota" y todas las subclases heredarán automáticamente este comportamiento.

Sin embargo, es importante tener en cuenta que la herencia puede llevar a una jerarquía de clases compleja y difícil de mantener, especialmente si se utilizan múltiples niveles de herencia. Además, la herencia puede introducir acoplamiento entre las clases, lo que significa que los cambios en una clase pueden afectar a otras clases relacionadas. Por lo tanto, es importante usar la herencia de manera cuidadosa y equilibrada en el diseño de nuestro sistema de gestión de mascotas.

### *Polimorfismo*

Otro concepto fundamental en la programación orientada a objetos es el polimorfismo, que se refiere a la capacidad de objetos de diferentes clases de responder de manera diferente a un mismo mensaje o método. El polimorfismo permite que un objeto pueda comportarse de diferentes maneras dependiendo del contexto en el que se utilice, lo que aumenta la flexibilidad y la reutilización de código.

Para entender el polimorfismo, consideremos un ejemplo de una clase Animal que tiene un método hacerSonido(). Esta clase puede ser heredada por otras clases, como Gato, Perro y Caballo, que también tienen su propio método hacerSonido(). Cada método de hacerSonido() es específico de la clase y define el sonido que hace el animal correspondiente.

Ahora imaginemos que tenemos un arreglo de animales, que puede contener instancias de las clases Gato, Perro y Caballo. Si queremos hacer que cada animal haga su sonido, podemos iterar a través del arreglo y llamar al método hacerSonido() para cada objeto en el arreglo. Aunque estamos llamando al mismo método para cada objeto, cada objeto responde de manera diferente, ya que cada clase tiene su propia implementación del método hacerSonido(). Esto es un ejemplo de polimorfismo en acción.

El polimorfismo es útil cuando queremos tratar a diferentes objetos de manera similar, pero aún así necesitamos que cada objeto responda de manera diferente. En lugar de tener que crear un método diferente para cada clase, podemos definir un único método que pueda ser llamado en diferentes objetos con diferentes implementaciones.

Además, el polimorfismo también es importante para la herencia, ya que nos permite crear clases hijas que extienden el comportamiento de la clase padre, pero también pueden agregar comportamientos específicos de la clase hija.

### *Cohesión y acoplamiento*

Cohesión y acoplamiento son conceptos fundamentales en la programación orientada a objetos que se refieren a la relación entre los objetos en un sistema y cómo interactúan entre sí. Ambos conceptos están estrechamente relacionados y son importantes para diseñar sistemas eficientes y mantenibles.

La cohesión se refiere a la medida en que los elementos de un módulo están relacionados y trabajan juntos para lograr una tarea específica. En otras palabras, la cohesión mide cuán enfocado y específico es un módulo en su propósito y cuánto sentido tiene agrupar esos elementos juntos.

La alta cohesión es deseable porque significa que los elementos de un módulo están altamente relacionados y trabajan juntos para lograr un propósito específico. Esto facilita la comprensión y mantenimiento del código, así como la reutilización de componentes. Por otro lado, la baja cohesión significa que los elementos de un módulo están poco relacionados y pueden realizar múltiples tareas, lo que puede hacer que sea más difícil entender su propósito y modificar su comportamiento.

Un ejemplo de alta cohesión podría ser una clase que se encarga exclusivamente de la gestión de un conjunto de objetos similares, como una clase de "Perros" que tiene métodos para agregar, eliminar, buscar y actualizar objetos de tipo "Perro". En cambio, un ejemplo de baja cohesión podría ser una clase que tiene métodos para manipular diferentes tipos de objetos, como una clase de "Animales" que tiene métodos para gestionar tanto perros como gatos.

Por otro lado, el acoplamiento se refiere a la medida en que los módulos dependen entre sí para funcionar correctamente. En otras palabras, el acoplamiento mide la interconexión entre los módulos de un sistema. Un acoplamiento alto significa que los módulos dependen mucho entre sí, lo que puede dificultar el mantenimiento y la modificación del código. Por otro lado, un acoplamiento bajo significa que los módulos son independientes y pueden modificarse sin afectar otros módulos en el sistema.

Un ejemplo de acoplamiento alto podría ser una clase que depende de múltiples clases para realizar su tarea, como una clase de "Veterinario" que depende de las clases de "Perros" y "Gatos" para realizar su tarea de cuidado de mascotas. En cambio, un ejemplo de acoplamiento bajo podría ser una clase que no depende de otras clases, como una clase de "Calculadora" que realiza operaciones matemáticas simples sin interactuar con otros módulos en el sistema.

Es importante lograr un equilibrio adecuado entre la cohesión y el acoplamiento para diseñar sistemas eficientes y mantenibles. La alta cohesión y el bajo acoplamiento son objetivos deseables porque significan que los módulos están altamente enfocados en su propósito y son independientes de otros módulos en el sistema. Esto facilita la comprensión y mantenimiento del código, así como la reutilización de componentes.

## Lenguajes de programación orientada a objetos

### *Comparación de distintos lenguajes O.O. (C++, Java, PHP, Ruby, TypeScript)*

La programación orientada a objetos es una metodología que se ha vuelto cada vez más popular en los últimos años. Con el aumento de la demanda de aplicaciones y sistemas de software cada vez más complejos, los desarrolladores han comenzado a adoptar lenguajes de programación orientados a objetos para facilitar el proceso de desarrollo.

*C++*

C++ es un lenguaje de programación orientado a objetos que ha existido desde la década de 1980. Se considera uno de los lenguajes más poderosos y eficientes en términos de velocidad y memoria. Esto se debe en gran parte a que C++ es un lenguaje compilado que permite al programador un control preciso sobre la asignación de memoria y los recursos del sistema.

C++ es un lenguaje complejo y puede resultar difícil de aprender para los principiantes. Sin embargo, debido a su eficiencia y capacidad para programar aplicaciones de bajo nivel, C++ se utiliza ampliamente en aplicaciones de gráficos de alta calidad, juegos y aplicaciones de control de dispositivos.

*Java*

Java es un lenguaje de programación orientado a objetos que se utiliza ampliamente en la creación de aplicaciones empresariales y en línea. Java es popular porque es un lenguaje portable y se puede ejecutar en cualquier plataforma. Java también tiene una gran biblioteca estándar que proporciona una amplia variedad de funcionalidades para la creación de aplicaciones.

Una de las ventajas de Java es su capacidad para manejar grandes proyectos de software de manera efectiva. Java también se utiliza en la programación de aplicaciones móviles y en la creación de aplicaciones de escritorio. Java es un lenguaje de programación fácil de aprender y cuenta con una gran comunidad de desarrolladores.

*PHP*

PHP es un lenguaje de programación orientado a objetos utilizado principalmente para la creación de aplicaciones web. PHP es fácil de aprender y ofrece una gran cantidad de características y funcionalidades que facilitan la creación de aplicaciones web dinámicas. PHP también se integra fácilmente con HTML, lo que permite a los desarrolladores crear aplicaciones web de manera rápida y sencilla.

Una de las desventajas de PHP es que no es tan rápido como otros lenguajes de programación orientada a objetos como C++ y Java. Sin embargo, PHP sigue siendo popular debido a su facilidad de uso y a la gran cantidad de recursos y bibliotecas disponibles para los desarrolladores.

*Ruby*

Ruby es un lenguaje de programación orientado a objetos que se ha vuelto cada vez más popular en los últimos años. Ruby se utiliza ampliamente en la creación de aplicaciones web y en la creación de prototipos de aplicaciones. Ruby es fácil de aprender y cuenta con una sintaxis sencilla y fácil de entender.

Una de las características más atractivas de Ruby es su capacidad para manejar código en tiempo de ejecución. Esto significa que los desarrolladores pueden agregar o modificar código en tiempo real, lo que facilita la creación de aplicaciones dinámicas y flexibles. Ruby también tiene una gran comunidad de desarrolladores y una gran cantidad de recursos disponibles.

*TypeScript*

TypeScript es un lenguaje de programación orientado a objetos que se basa en JavaScript, pero incluye algunas características adicionales. Una de las principales ventajas de TypeScript es que es más seguro que JavaScript, ya que detecta errores en tiempo de compilación en lugar de en tiempo de ejecución. Esto significa que los desarrolladores pueden encontrar errores en su código antes de que el programa se ejecute, lo que ahorra tiempo y evita posibles problemas en producción.

TypeScript también ofrece una sintaxis más limpia y legible que JavaScript, lo que hace que el código sea más fácil de entender y mantener. Además, TypeScript incluye funciones de programación orientada a objetos como clases, interfaces y herencia, lo que facilita la creación de aplicaciones escalables y de alta calidad.

*C++:*

```
#include <iostream>
using namespace std;

class Animal {
    public:
        virtual void hacerSonido() {
```

```
            cout << "Sonido genérico de animal" << endl;
        }
};

class Perro : public Animal {
    public:
        void hacerSonido() {
            cout << "Guau! Guau!" << endl;
        }
};

int main() {
    Animal* animal = new Animal();
    Perro* perro = new Perro();

    animal->hacerSonido();
    perro->hacerSonido();

    return 0;
}
```

*Java:*

```
class Animal {
    public void hacerSonido() {
        System.out.println("Sonido genérico de animal");
    }
}

class Perro extends Animal {
    public void hacerSonido() {
        System.out.println("Guau! Guau!");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Perro perro = new Perro();

        animal.hacerSonido();
        perro.hacerSonido();
    }
}
```

*PHP:*

```
<?php
class Animal {
    public function hacerSonido() {
        echo "Sonido genérico de animal\n";
    }
}

class Perro extends Animal {
    public function hacerSonido() {
```

```
        echo "Guau! Guau!\n";
    }
}

$animal = new Animal();
$perro = new Perro();

$animal->hacerSonido();
$perro->hacerSonido();
?>
```

*Ruby:*

```
class Animal
    def hacerSonido
        puts "Sonido genérico de animal"
    end
end

class Perro < Animal
    def hacerSonido
        puts "Guau! Guau!"
    end
end

animal = Animal.new
perro = Perro.new

animal.hacerSonido
perro.hacerSonido
```

*TypeScript:*

```
class Animal {
    hacerSonido() {
        console.log("Sonido genérico de animal");
    }
}

class Perro extends Animal {
    hacerSonido() {
        console.log("Guau! Guau!");
    }
}

const animal = new Animal();
const perro = new Perro();

animal.hacerSonido();
perro.hacerSonido();
```

En conjunto, los códigos de los diferentes lenguajes implementan los conceptos fundamentales de la programación orientada a objetos como la abstracción, encapsulamiento, herencia y polimorfismo en el contexto de la clase Perro y la clase Animal.

En términos generales, se puede concluir que todos los lenguajes de programación orientada a objetos tienen similitudes en la forma en que se implementan los conceptos mencionados anteriormente. Sin embargo, cada lenguaje tiene sus propias características y ventajas.

En cuanto a las coincidencias en sintaxis y parecidos en la implementación, se pueden observar algunas similitudes entre los lenguajes estudiados. Por ejemplo, en cuanto a la implementación del concepto de Abstracción, se puede ver que en todos los lenguajes se utilizan clases para definir objetos que encapsulan datos y comportamientos. En cuanto al concepto de Encapsulamiento, se utiliza una sintaxis similar en todos los lenguajes para definir los atributos y métodos privados y públicos de una clase.

En el concepto de Herencia, se observa que todos los lenguajes tienen un mecanismo para heredar propiedades y comportamientos de una clase a otra, a través de una jerarquía de clases. Y en el concepto de Polimorfismo, se puede observar que todos los lenguajes tienen la capacidad de definir métodos con el mismo nombre en diferentes clases, y que se comportan de manera diferente según la clase a la que pertenecen.

Sin embargo, también hay diferencias significativas en la sintaxis y en la implementación de los conceptos de POO entre los diferentes lenguajes. Por ejemplo, en Java y TypeScript se utiliza la palabra clave "extends" para definir una clase hija, mientras que en C++ se utiliza el operador de dos puntos "::". Además, en Ruby y PHP no hay una distinción clara entre los atributos y los métodos de una clase, mientras que en C++ y Java se utilizan modificadores de acceso para definirlos.

En conclusión, aunque todos los lenguajes de programación orientada a objetos comparten los mismos conceptos fundamentales, cada uno tiene su propia sintaxis y su propia implementación de estos conceptos. Es importante entender estas diferencias para poder utilizar de manera efectiva cada lenguaje y aprovechar al máximo sus características únicas.

## *Características del lenguaje Java.*

Java es un lenguaje de programación orientado a objetos muy utilizado en la actualidad, especialmente en el desarrollo de aplicaciones empresariales y móviles. Entre sus principales características, destacan las siguientes:

1. Portabilidad: Java se diseñó para ser portátil, lo que significa que los programas escritos en Java pueden ejecutarse en cualquier plataforma que tenga una máquina virtual Java (JVM) instalada. Esto permite que los programas escritos en Java se ejecuten en diferentes sistemas operativos sin tener que reescribir el código.

   Ejemplo: Un programa en Java escrito en Windows puede ejecutarse en Linux o Mac OS sin necesidad de realizar cambios en el código.

2. Robustez: Java se diseñó para ser un lenguaje robusto y resistente a errores, lo que significa que es menos propenso a fallas y errores en tiempo de ejecución. Esto se debe a que Java tiene un sistema de gestión de memoria automático que controla la asignación y liberación de memoria, evitando así errores comunes como desbordamiento de memoria y pérdida de memoria.

   Ejemplo: En Java, no es necesario liberar manualmente la memoria asignada a una variable. El sistema de gestión de memoria se encarga de esto automáticamente.

3. Orientado a objetos: Java es un lenguaje de programación orientado a objetos, lo que significa que se basa en el concepto de objetos y clases. Los objetos son instancias de una clase y se utilizan para representar entidades del mundo real.

   Ejemplo: En Java, se puede crear una clase "Perro" que tenga atributos como "raza", "edad" y "color". A partir de esta clase, se pueden crear múltiples objetos "Perro" con diferentes valores para estos atributos.

4. Seguridad: Java se diseñó con la seguridad en mente, lo que significa que está diseñado para evitar errores de programación y para proteger contra ataques maliciosos. Java tiene una serie de características de seguridad incorporadas, como la verificación de tipos, la gestión de excepciones y la gestión de permisos.

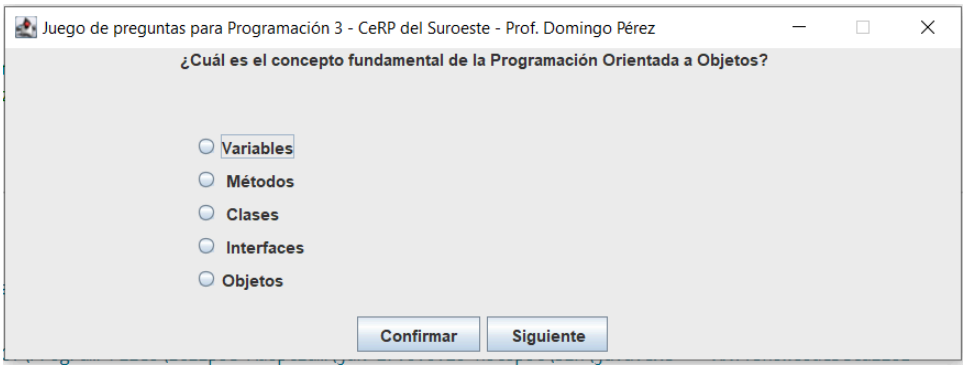
   Ejemplo: En Java, si se intenta acceder a un objeto que no tiene permiso para acceder, se lanzará una excepción de seguridad.

5. Multi-hilo: Java admite la programación multi-hilo, lo que significa que se pueden crear programas que realicen varias tareas simultáneamente. Java proporciona una serie de funciones y herramientas para gestionar los hilos de forma segura y eficiente.

   Ejemplo: En Java, se puede crear un programa que realice varias tareas simultáneamente, como descargar un archivo mientras se actualiza una base de datos en segundo plano.

## Actividades

1. Te invito nuevamente a ingresar a github.com/Domingo1987/Curso-JAVA y buscar el proyecto de Java llamado "PreguntasDelCursoMVC". Este proyecto es de gran importancia ya que te permitirá no solo evaluar tus aprendizajes sino también comenzar a ver algunos conceptos de los que trabajaremos en el curso. Puedes clonarlo y trabajar en él para mejorar tus habilidades en programación en Java.




2. Bienvenidos al proyecto "POOShowcase" en el curso de Programación 2 con Java. Este proyecto tiene como finalidad el desarrollo de una aplicación que permita al usuario visualizar un tema, su descripción detallada, y acceder a un enlace directo a un código en Replit que ilustre su aplicación práctica. Para dar inicio a este desafío, es necesario que clonen el repositorio "PreguntasDelCursoMVC" disponible en la página de GitHub del curso, donde encontrarán la base necesaria para comenzar a trabajar en la aplicación.

   La estructura del proyecto incluye varias clases, algunas de ellas ya están implementadas para guiarlos en el proceso, mientras que otras requerirán de su ingenio y habilidades de programación para ser completadas o adaptadas a los requisitos del proyecto.

   Para interactuar con la aplicación, el usuario seleccionará un tema de interés y, como respuesta, el sistema le presentará una descripción extensa del mismo, junto con un enlace a Replit donde podrán ver y experimentar con un código que demuestra cómo implementar el concepto seleccionado en un contexto real. El propósito de este proyecto es que logren finalizar el desarrollo de la aplicación, apoyándose en la documentación proporcionada en el sitio web del curso en GitHub y los recursos adicionales disponibles.

   ¡Manos a la obra! Con su habilidad y esfuerzo, lograrán culminar este proyecto con éxito y podrán poner en práctica los conceptos fundamentales de la POO. Abstracción, Encapsulamiento, Herencia y Polimorfismo, utilizando el lenguaje de programación Java.