POO
Parte I

INSTITUTO DE INNOVACIÓN Y
TECNOLOGÍA APLICADA

IITA

Contenido

¿Qué es el POO? ¿Por qué nacio?

PREGUNTA INICIAL

BREVE REPASO

Clase

Objeto

```python
class Perro:
    # Constructor (inicializa los atributos)
    def __init__(self, nombre, raza, edad):
        self.nombre = nombre  # Atributo de la clase
        self.raza = raza  # Atributo de la clase
        self.edad = edad  # Atributo de la clase

    # Método que hace que el perro ladre
    def ladrar(self):
        print(f"{self.nombre} dice: ¡Guau!")

    # Método que hace que el perro camine
    def caminar(self):
        print(f"{self.nombre} está caminando...")
```

Métodos

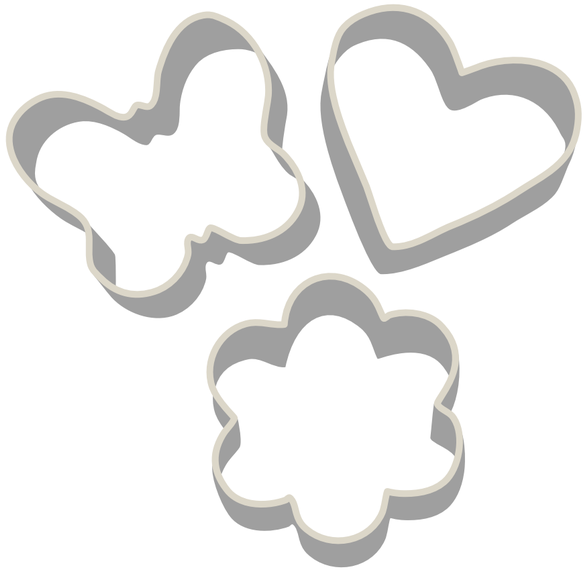
Atributos

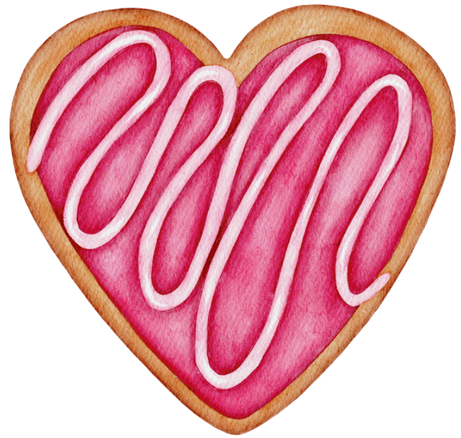
La **Programación Orientada a Objetos (POO)** es un paradigma de programación que organiza el código en "objetos" que contienen datos y funciones. La POO nació como respuesta a la creciente complejidad del software, permitiendo agrupar datos y comportamientos de manera más eficiente.

¿Diferencia entre una Clase y un Objeto?

PREGUNTA DE ENTREVISTA

BREVE REPASO





Clases

Objetos

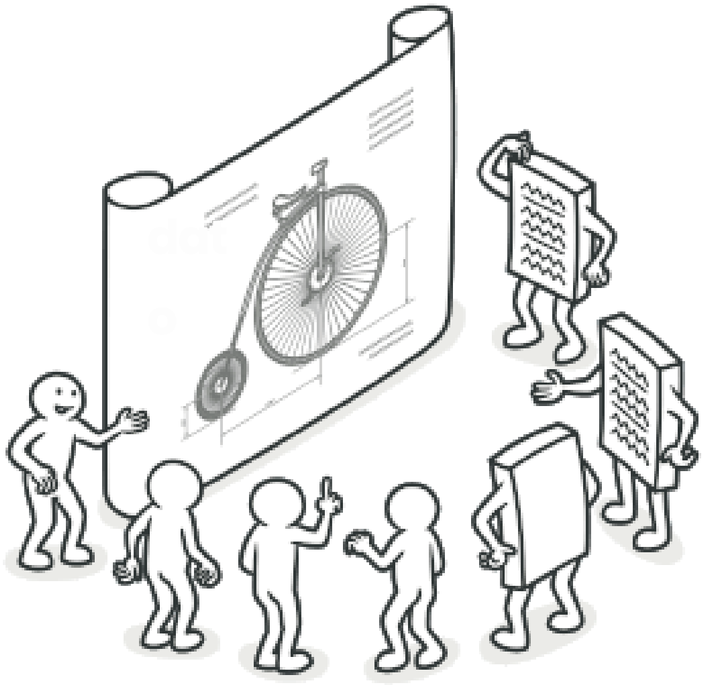
Una **clase** es el plano o molde del **objeto**. Es como la definición de una "nave" o "perro". En cambio, un **objeto** una instancia de una **clase**. Cuando creas una nave específica (como "Nave 1") usando la clase "Nave", estás creando un objeto.

Cuales son los 6 principios del POO?

PREGUNTA DE ENTREVISTA

6 PRINCIPIOS DE POO

- **Herencia**
- **Cohesión**
- **Abstracción**



- **Polimorfismo**
- **Acoplamiento**
- **Encapsulamiento**

Estos principios permiten escribir código más **estructurado, reutilizable y fácil de mantener**.

Herencia

: Permite que una clase **herede** atributos y métodos de otra clase.

Esto facilita la **reutilización** del código y la **extensión** de funcionalidades sin tener que reescribirlo.

HERENCIA

Clase padre que **tiene las cosas comunes para todos.**

```java
// Animal is a Parent class
class Animal
{
    public void eat()
    {
        System.out.println("Animal is eating");
    }
}

// Dog is derived from Animal class
class Dog extends Animal
{
    public static void main(String args[])
    {
        // Creating object of Dog class
        Dog d = new Dog();

        // Dog can access eat() method
          // of Animal class
        d.eat();
    }
}
```

JAVA

Clase hija que solo **reutiliza**

```python
# Clase padre: Animal
class Animal:
    def __init__(self, especie, edad):
        self.especie = especie
        self.edad = edad

    def describeme(self):
        print(f"Soy un {self.especie} y tengo {self.edad} años.")

# Clase hija: Perro (hereda de Animal)
class Perro(Animal):
    pass

# Crear un perro
mi_perro = Perro("mamífero", 5)
mi_perro.describeme()  # Salida: Soy un mamífero y tengo 5 años.
```

Python

La **Herencia** nos sirve para crear algo nuevo (una "**clase hija**") basado en algo ya existente (una "**clase padre**").

¿Para qué sirve la herencia?

PREGUNTA DE ENTREVISTA

HERENCIA

```python
# Clase hija: Vaca
class Vaca(Animal):
    pass

# Clase hija: Abeja
class Abeja(Animal):
    pass

# Crear una vaca y una abeja
mi_vaca = Vaca("mamífero", 7)
mi_abeja = Abeja("insecto", 1)

mi_vaca.describeme()  # Salida: Soy un mamífero y tengo 7 años.
mi_abeja.describeme()  # Salida: Soy un insecto y tengo 1 años.
```

La herencia nos ayuda a **reutilizar** código y evitar **repetirnos**. Por ejemplo, si queremos hacer más animales (como vacas o abejas), no tenemos que escribir todo desde cero.

HERENCIA

```python
class Perro(Animal):
    def hablar(self):
        print("¡Guau!")

class Vaca(Animal):
    def hablar(self):
        print("¡Muuu!")

class Abeja(Animal):
    def hablar(self):
        print("¡Bzzz!")

# Crear animales
mi_perro = Perro("mamífero", 5)
mi_vaca = Vaca("mamífero", 7)
mi_abeja = Abeja("insecto", 1)

mi_perro.hablar()  # Salida: ¡Guau!
mi_vaca.hablar()   # Salida: ¡Muuu!
mi_abeja.hablar()  # Salida: ¡Bzzz!
```

A veces queremos que las **clases hijas** hagan las cosas de forma diferente. Por ejemplo, cada animal puede "hablar" de una manera distinta. Esto lo logramos **sobrescribiendo** (modificando) métodos en las clases hijas.

¿Como se puede acceder al constructor de la clase padre?

PREGUNTA DE ENTREVISTA

HERENCIA

```python
class Perro(Animal):
    def __init__(self, especie, edad, dueño):
        super().__init__(especie, edad)  # Llamamos al constructor de Animal
        self.dueño = dueño  # Agregamos algo nuevo

mi_perro = Perro("mamífero", 3, "Luis")
print(f"Especie: {mi_perro.especie}, Edad: {mi_perro.edad}, Dueño: {mi_perro.dueño}")
# Salida: Especie: mamífero, Edad: 3, Dueño: Luis
```

Cuando creamos una clase hija, a veces queremos usar el constructor de la clase padre y agregar algo más. Para eso usamos **super()**.

¿Qué es la herencia multiple?

PREGUNTA DE ENTREVISTA

HERENCIA

```python
class Mamifero:
    def amamantar(self):
        print("Dando leche...")

class Volador:
    def volar(self):
        print("Volando alto...")

# Clase hija que hereda de dos padres
class Murcielago(Mamifero, Volador):
    pass

mi_murcielago = Murcielago()
mi_murcielago.amamantar()  # Salida: Dando leche...
mi_murcielago.volar()      # Salida: Volando alto...
```

Python también permite que una clase herede de varias clases padre. Esto se llama herencia múltiple.

Cohesión

La **cohesión** en programación se refiere a qué tan bien se relacionan entre sí los elementos dentro de una función o módulo. La idea es que los elementos de un módulo o función deben trabajar juntos para realizar una única tarea de manera **clara y eficiente.**

COHESIÓN

Cohesión débil

Cuando los elementos dentro de una función o módulo no están relacionados entre sí. Es decir, la función hace varias tareas que no tienen conexión entre ellas.

VS

Cohesión Fuerte

Es lo que buscamos. Indica que todos los elementos dentro de la función o módulo **están estrechamente relacionados** entre sí y cumplen con una única tarea bien definida.

Abstracción

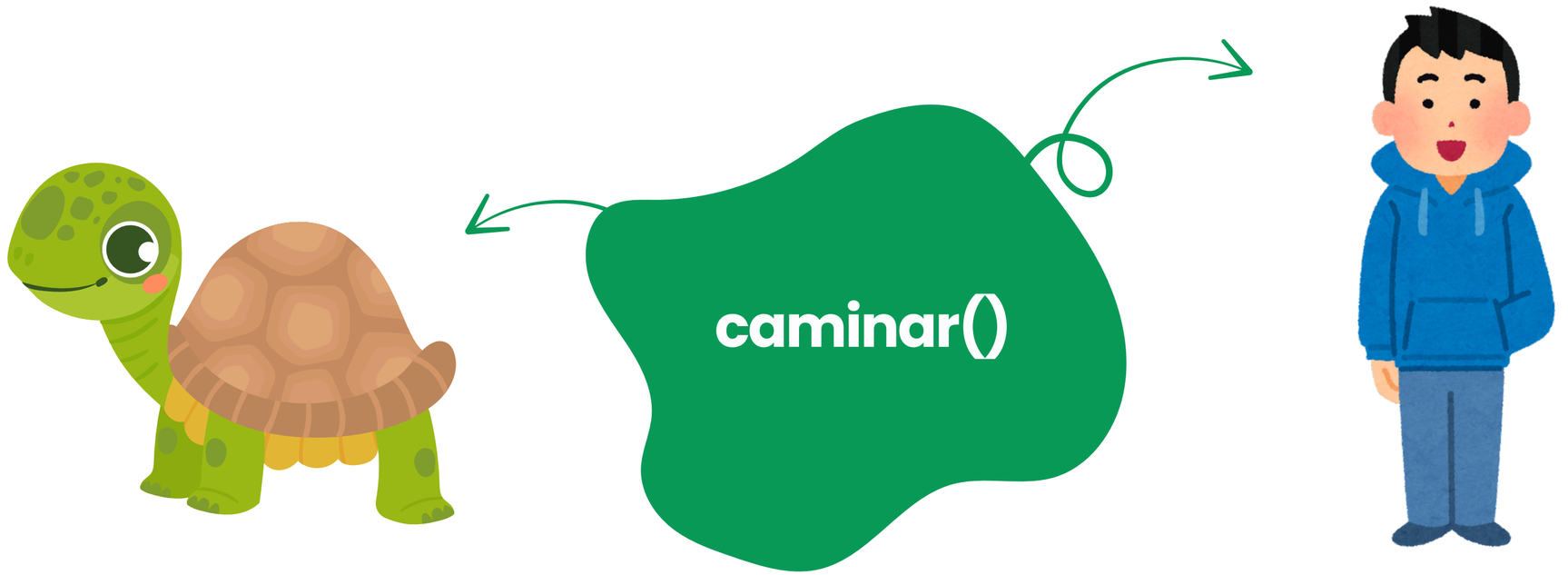
Se refiere a **ocultar los detalles complejos** de cómo funciona una aplicación o sistema, y enfocarse solo en lo que se necesita hacer o cómo se va a usar. Es como si tuvieras una herramienta compleja, pero **solo necesitas saber qué hace y no cómo lo hace**.

6 PRINCIPIOS DE POO

Polimorfismo

Es la : la capacidad de los objetos de asumir múltiples formas o comportamientos dependiendo del contexto.

POLIMORFISMO



El **polimorfismo** permite que objetos de clases diferentes sean tratados como si pertenecieran a una clase base común.

POLIMORFISMO

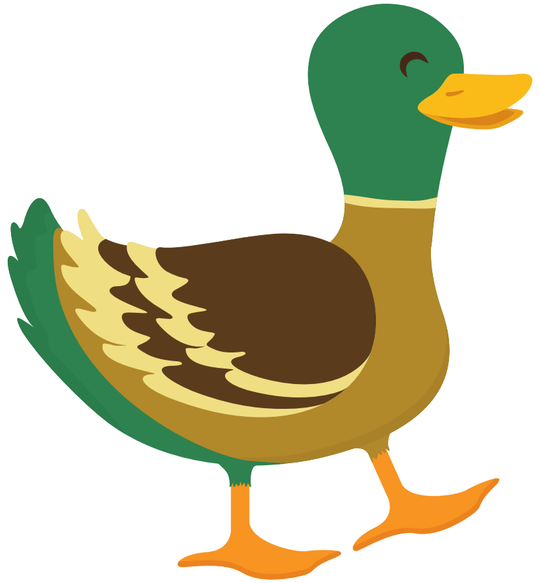
```python
class Animal:
    def hablar(self):
        pass


class Perro(Animal):
    def hablar(self):
        print("Guau!")

class Gato(Animal):
    def hablar(self):
        print("Miau!")
```

En Python, el polimorfismo es aún más flexible. Aquí **no se requiere** que las **clases compartan explícitamente una herencia común**; basta con que los objetos **implementen los métodos que se necesitan**. Este concepto se relaciona con el **duck typing**.

¿Qué es Duck typing?

PREGUNTA DE ENTREVISTA

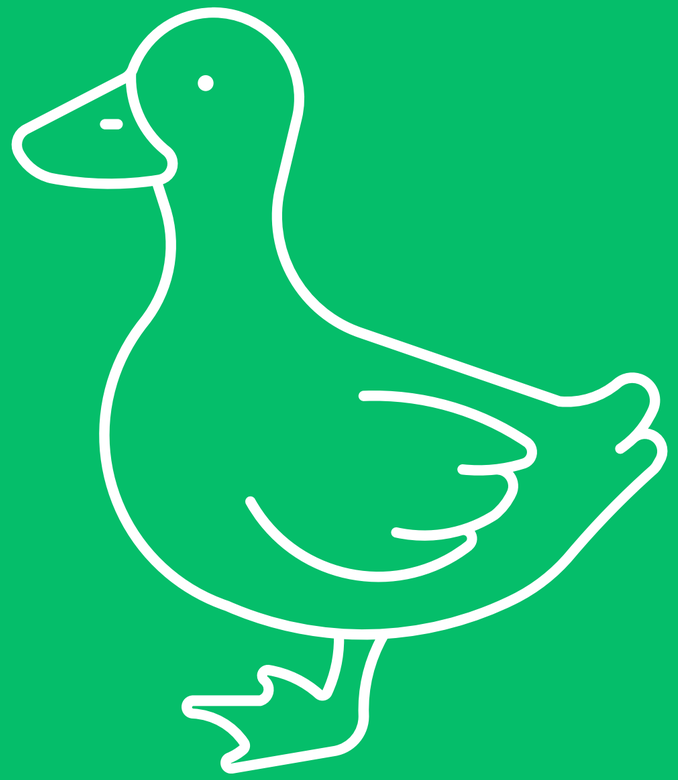
POLIMORFISMO



"Si camina como un pato y suena como un pato, entonces es un pato."

Significa que **no importa el tipo de un objeto, sino** que lo importante es **qué puede hacer**. Es decir, si un objeto tiene los **métodos o comportamientos** que necesitamos, entonces podemos tratarlo como si fuera del tipo que esperamos

¡Veamos el pato en acción!



Acoplamiento

Mide el **grado de dependencia** entre dos módulos de software, como **clases o componentes.** Este concepto es fundamental para construir sistemas escalables, mantenibles y robustos

ACOPLAMIENTO

Acoplamiento debil

Indica que los módulos tienen poca o ninguna dependencia entre sí.
Aumenta la modularidad y facilita la reutilización y el mantenimiento del código.

VS

Acoplamiento Fuerte

Sucede cuando los módulos dependen significativamente entre sí.
Genera rigidez y puede complicar el mantenimiento del sistema

Encapsulamiento

El encapsulamiento es como poner cosas privadas dentro de una caja que nadie puede abrir directamente. **Solo quien tiene la llave** (en este caso, la propia clase) **puede acceder** a lo que hay dentro de esa caja.

Veamoslo en Código...

PREGUNTA DE ENTREVISTA

IITA

¡Muchas Gracias!

Por su atención