Intro a POO

DEV.F:
DESARROLLAMOS(PERSONAS);

La estructura de un objeto literal está limitada por llaves, donde se encapsula cada identificador asignándole un valor literal, en un formato clave : valor. Si vamos a hacer uso del objeto más adelante, podemos almacenarlo en una variable asignándoselo con normalidad.

```
var nombreObjeto = {
    identificador1: valor1,
    identificador2: valor2,
    identificador_n: valor_n
}
```

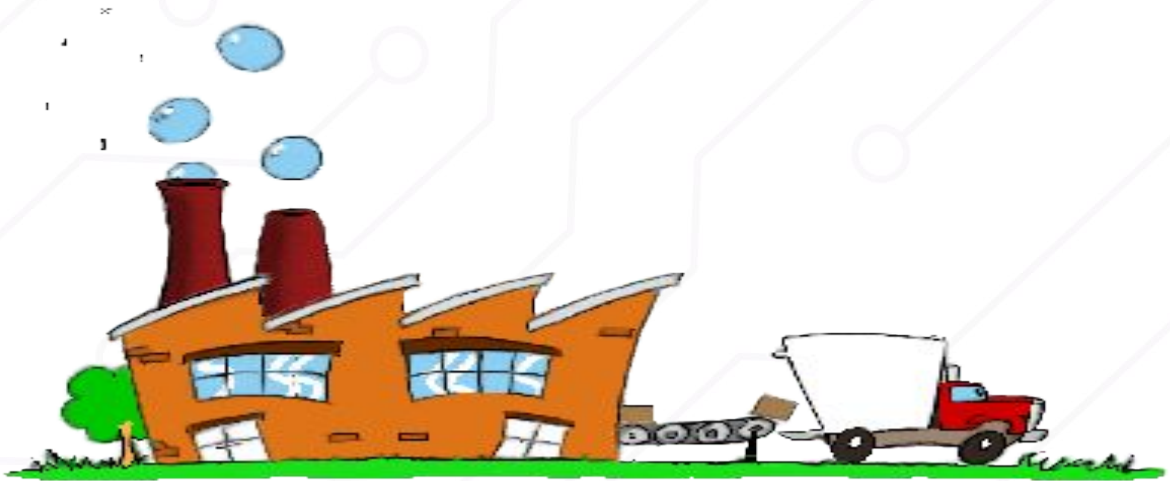
Todo como un objeto

En JavaScript, casi todo es un objeto. Todos los tipos primitivos excepto null y undefined se tratan como objetos. Pueden asignar propiedades, y tienen todas las características de los objetos.

Constructor

El constructor es un método especial que se ejecuta automáticamente cuando se crea una instancia de esa clase.

Su función es inicializar el objeto y sirve para asegurarnos que los objetos siempre contengan valores iniciales válidos.



Plasmar la abstracción en Código

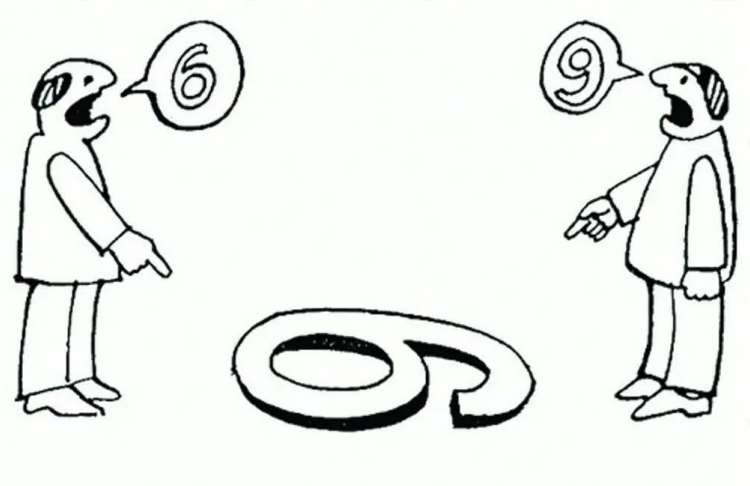
```
// ALUMNOS
class Alumnos {

    constructor( nombre, apellido, califFinal, inscrito){
        this.nombre = nombre;
        this.apellido = apellido;
        this.califFinal = califFinal;
        this.inscrito = inscrito;
    }
```

```
class persona {
    constructor(nombre){
       this.nombre = nombre;
    }

    function saludar(){
        return `hola ${this.nombre}!`;
    }
}
```

```
function persona(nombre){
    this.nombre = nombre;
}

persona.prototype.saludar = function (){
    return `hola ${this.nombre}!`;
}
```

¿Qué es un paradigma?

Normas que establecen límites y determinan cómo debe comportarse un elemento dentro de estos límites.

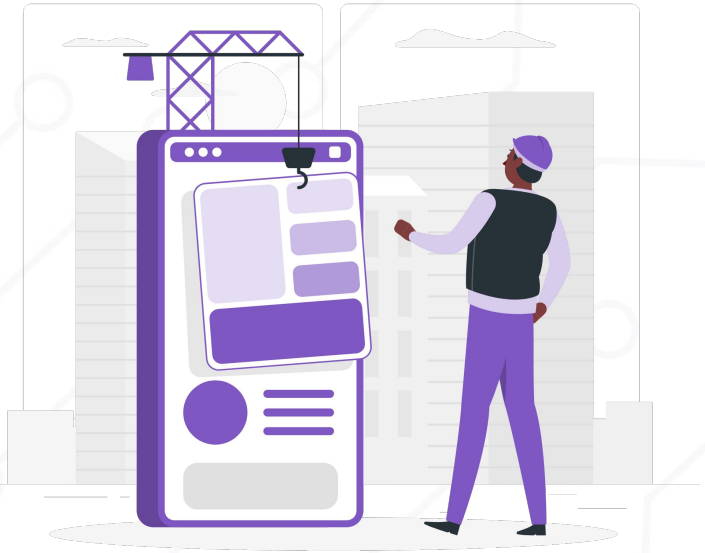


DEV.F:

Programación orientada a objetos

Intro a la programación orientada a objetos (POO)

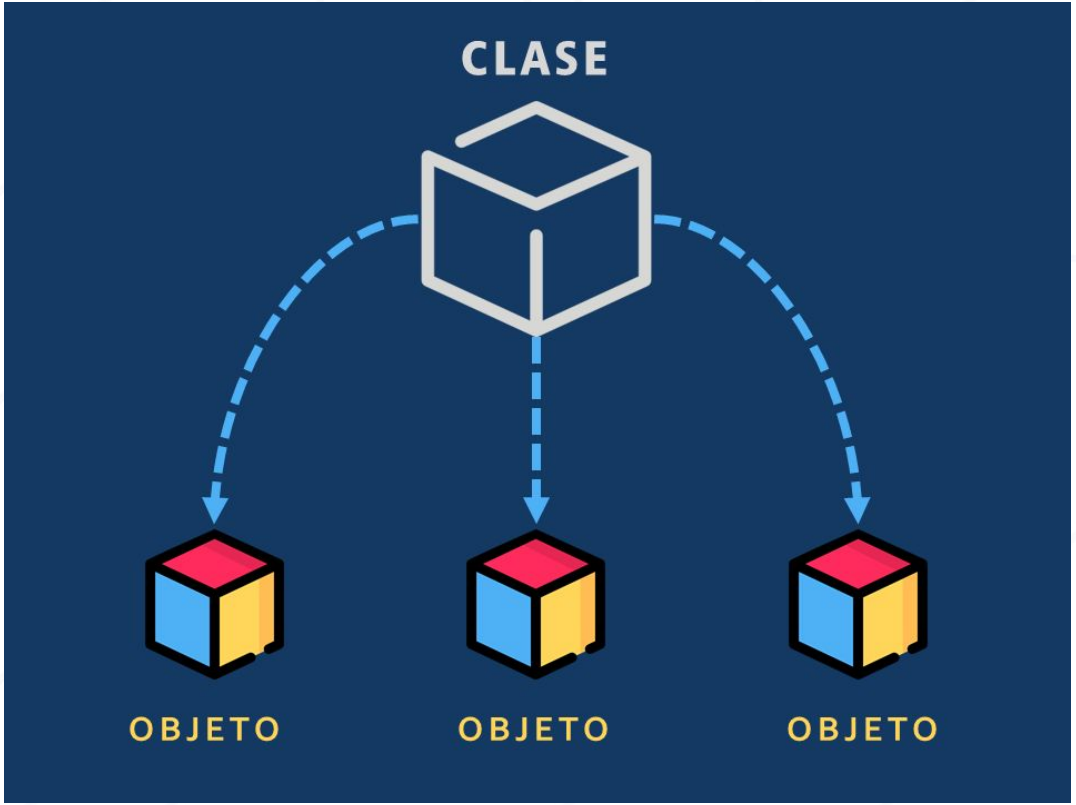
La programación Orientada a Objetos, también llamada POO, se define como un paradigma de programación, con el cual podemos:



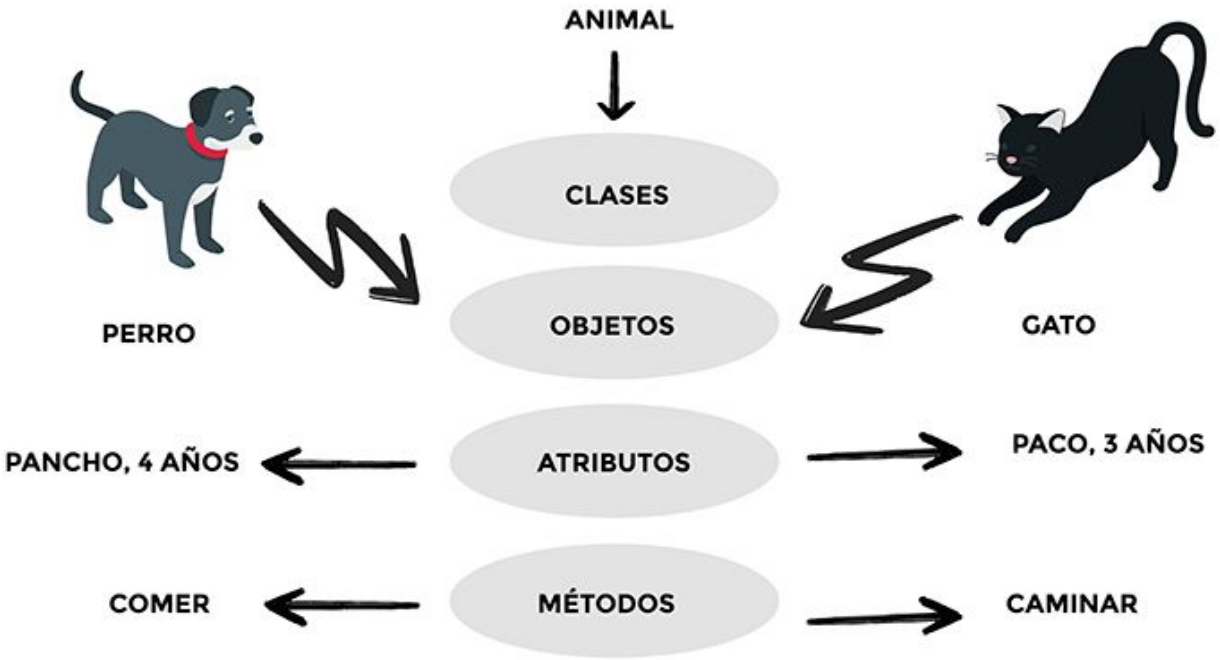
1. Invocar la manera en que se consiguen los resultados.
2. La programación orientada a objetos disminuye los errores y promociona la reutilización del código.
3. Es una manera especial de programar, que se acerca de alguna manera cómo podemos expresar las cosas en la vida real

Receta para entender POO

Los objetos se crean a partir de una plantilla llamada clase, cada objeto es una instancia de su clase

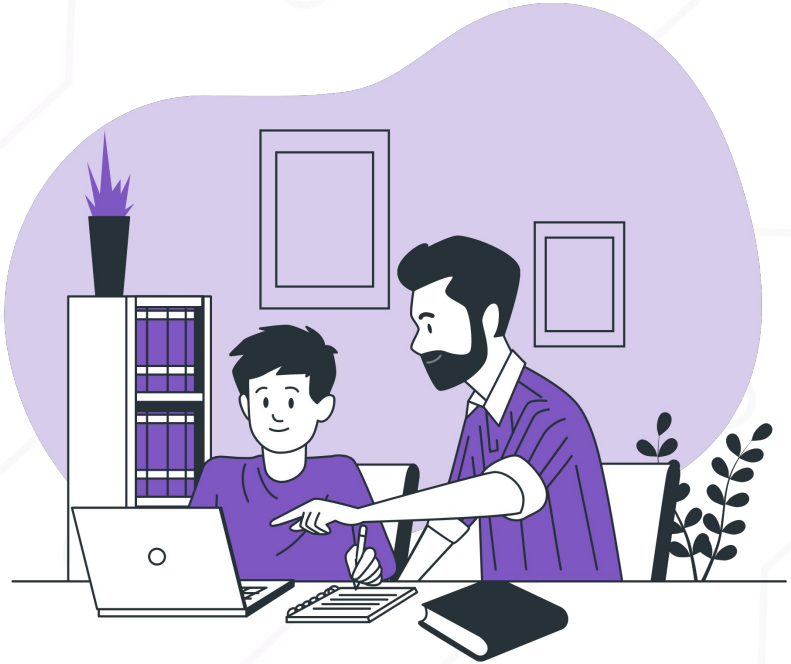


Descripción Gráfica



PARA QUE USAMOS POO

La idea básica de la Programación Orientada a Objetos (POO) es que usamos objetos para modelar cosas del mundo real que queremos representar en nuestros programas.

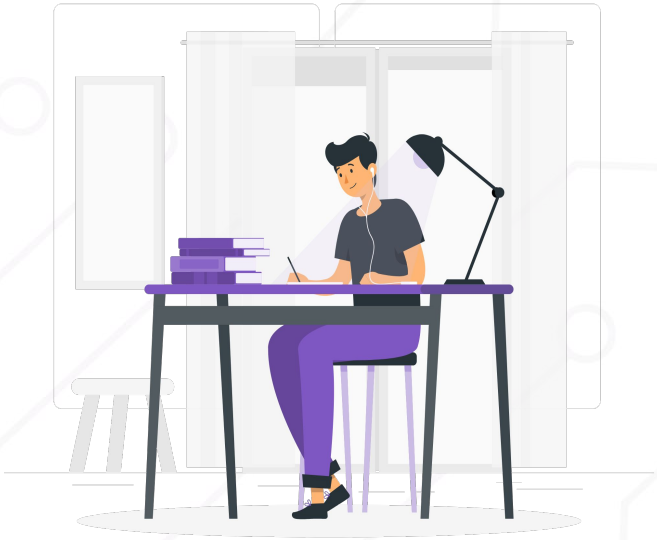


Información que cura

Los objetos pueden contener información y código relacionados, los cuales representan información acerca de lo que estás tratando de modelar, y la funcionalidad o comportamiento que deseas que tenga.

Los datos de un Objeto (y frecuentemente, también las funciones) se pueden almacenar ordenadamente (la palabra oficial es encapsular).

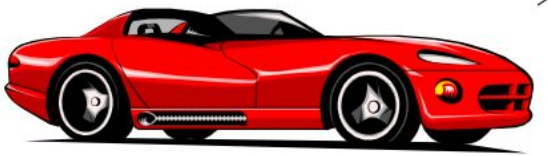
Los objetos también se usan comúnmente como almacenes de datos que se pueden enviar fácilmente a través de la red.



Pilares de la POO

Abstracción del objeto

Definiciones de las propiedades y comportamiento de un tipo de objeto concreto.



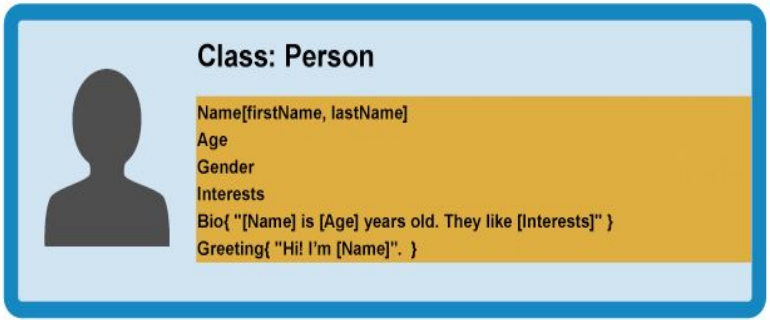
- Atributos:
  - color
  - velocidad
  - ruedas
  - motor
- Métodos:
  - arranca()
  - frena()
  - dobla()

Abstracción Visual

Abstracción

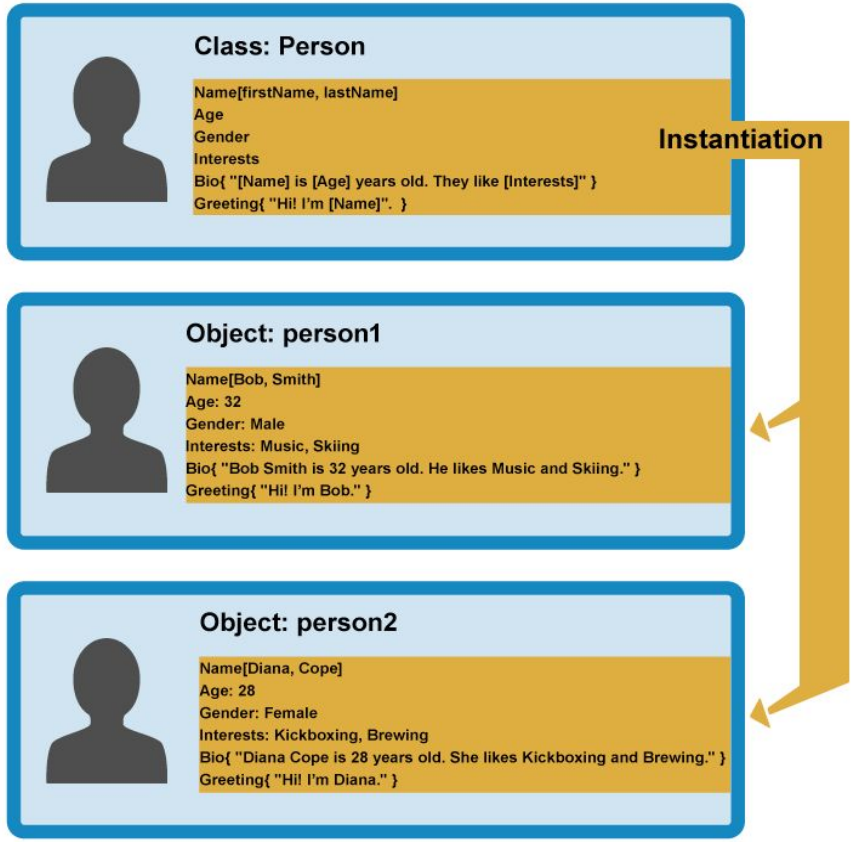
En este caso, solo estamos interesados en mostrar su nombre, apellido, calificación final y si está cursando el semestre, además de una pequeña introducción sobre este individuo basada en los datos anteriores.

Esto es conocido como abstracción — crear un modelo simple de algo complejo que represente sus aspectos más importantes y que sea fácil de manipular para el propósito de nuestro programa.



Creando Objetos

Partiendo de nuestra clase, podemos crear instancias de objetos — objetos que contienen los datos y funcionalidades definidas en la clase original. Teniendo a nuestra clase `Person`, ahora podemos crear gente con características más específicas:



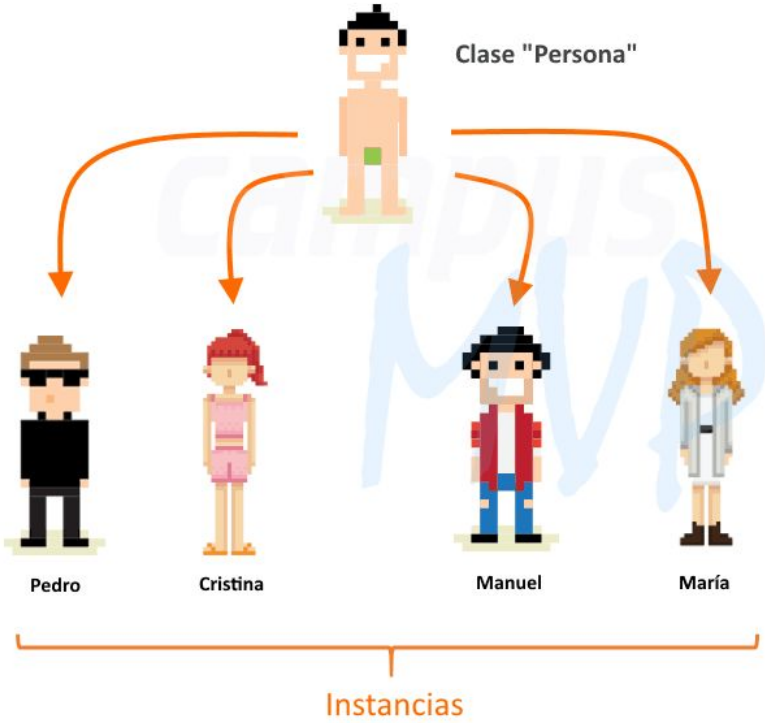
Abstracción

- Debe enfocarse a lo mínimo.
- Se busca definir atributos y métodos más relevantes.
- Eventualmente como programadores desarrollamos la capacidad de abstracción.

ENCAPSULAMIENTO

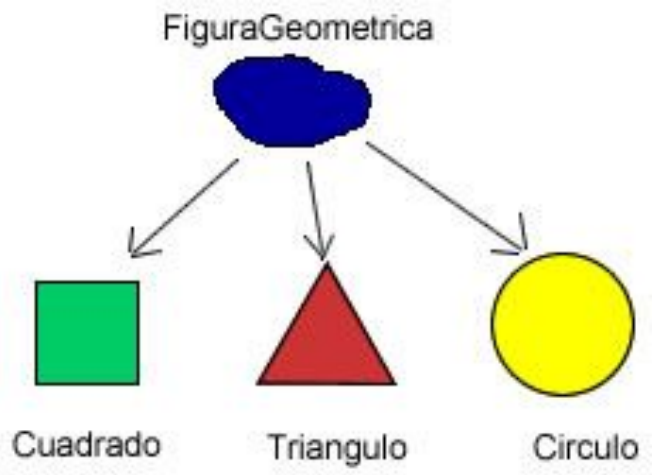
El encapsulamiento es un concepto que nos permite proteger el estado interno de nuestros objetos para que no pueda ser accedido y modificado por cualquiera.

Podemos definir la privacidad de los datos y solo permitir que se modifiquen los que exponemos. Por ejemplo, si tenemos un método llamado 'cambiarCollar' que nos pemita cambiar el color del cascabel del gato se podría acceder y cambiar esta información sin tocar otros atributos como el peso o la edad



Encapsulamiento

- Hablamos de agrupamiento y protección.
- Colocar atributos y métodos en un mismo lugar (Clase)
- Se busca lograr que un objeto no revele los datos de sí mismo a menos que sea necesario.



Polimorfismo

- Se utiliza cuando una clase hereda sus atributos y métodos.

- Sobreescritura de métodos.

DEV.F::

Herencia

EN PROGRAMACIÓN LA HERENCIA ES LA CAPACIDAD DE PASAR SUS CARACTERÍSTICAS(TANTO ATRIBUTOS COMO MÉTODOS) DE UNA CLASE A OTRA.

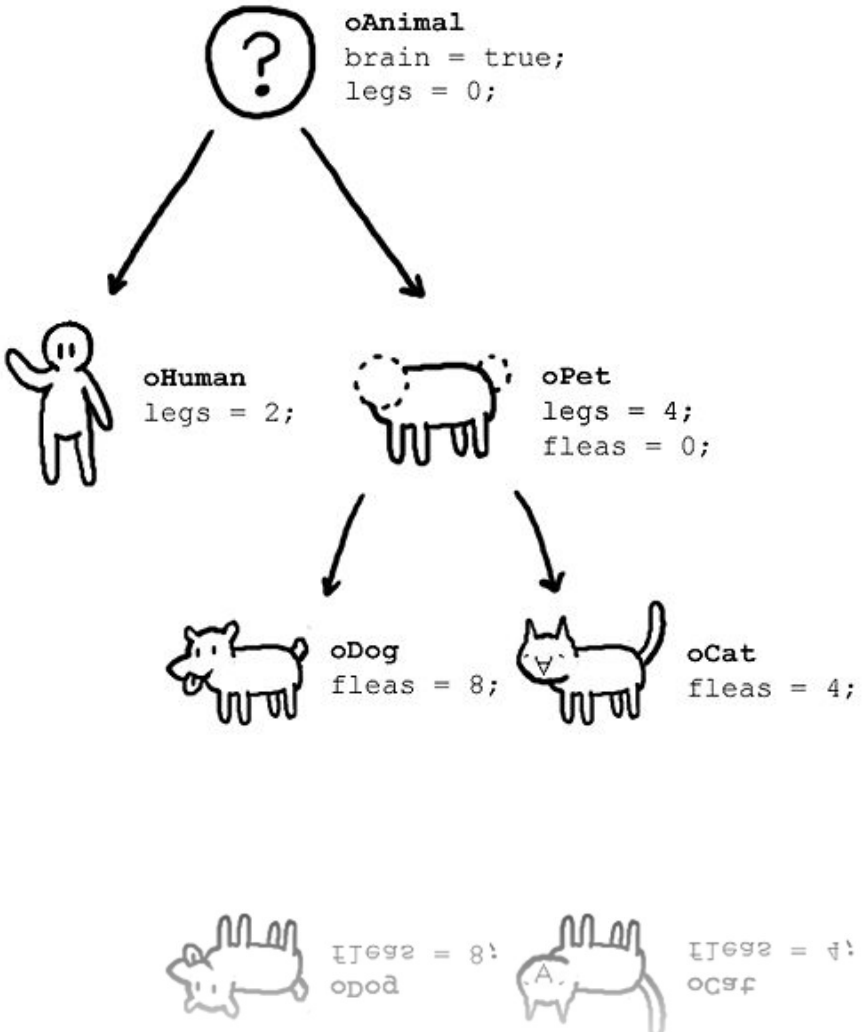
Otra ventaja de la herencia es la capacidad para definir atributos y métodos nuevos para la subclase.

CLASES Y SUBCLASES

CLASE --> PADRE O SUPERCLASE
SUBCLASE --> HIJO

POR EJEMPLO PODRÍAMOS TENER UNA CLASE “MAMIFERO” QUE TENGA CIERTOS ATRIBUTOS COMO "PELO", "OJOS", "OREJAS". TANTO LA SUBCLASE GATITO COMO LA SUBCLASE PERRITO, PODRÍAN HEREDAR DE “MAMIFERO”.

NOTA: La herencia realiza la relación es-un
Un gatito es-un mamifero; un perro es-un mamífero, etc.



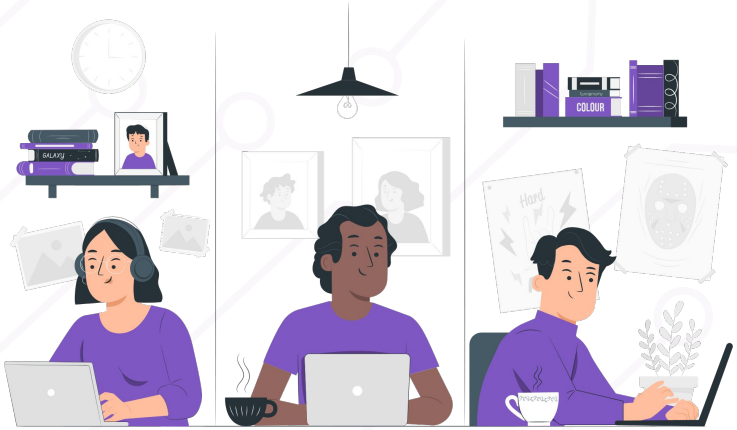
Herencia

- Crear una clase a partir de una existente.
- Se heredan atributos y métodos.

DEV.F::

EN UNA DEFINICIÓN MÁS TÉCNICA HERENCIA.

- Es un mecanismo para la reutilización de software.
- Permite definir a partir de una clase otras clases relacionadas a mi superclase.



Ejemplo

En un cine se reproducen largometrajes. Puedes, no obstante, tener varios tipos de largometrajes, como películas, documentales, etc.

Quizá las películas y documentales tienen diferentes características, distintos horarios de audiencia, distintos precios para los espectadores y por ello has decidido que tu clase "Largometraje" tenga clases hijas o derivadas como "Película" y "Documental".

Imagina que en tu clase "Cine" creas un método que se llama "reproducir()".

Este método podrá recibir como parámetro aquello que quieres emitir en una sala de cine y podrán llegarte a veces objetos de la clase "Película" y otras veces objetos de la clase "Documental".

Si quisiera reproducir una película tendría los siguiente:

reproducirPelicula(peliculaParaReproducir){... }

Pero si luego tienes que reproducir documentales, tendrás que declarar:

reproducirDocumental(documentaParaReproducir){... }

¿Realmente es necesario hacer dos métodos?