

# Objetivo de la sesión

- En la sesión de hoy veremos Herencia tema relacionado a las Clases en Javascript
- Veremos la parte teórica y pasaremos a la práctica con un ejemplo
- Después se dividirán en salas para trabajar dos ejercicios por equipos, lo que resta de la clase.



*¡QUE DISFRUTES LA SESIÓN!*

# Clases y Objetos

EN PROGRAMACIÓN ORIENTADA A  
OBJETOS ES IMPORTANTE  
RECORDAR QUE UTILIZAMOS **CLASES**  
Y **OBJETOS** DONDE  
**UNA CLASE SE DECLARA Y**  
**UN OBJETO SE CREA**

Una **clase** es una entidad abstracta

- Es un tipo de clasificación de datos
- Define el comportamiento y atributos de un grupo de estructura y comportamiento similar

Clase Coche

Métodos: arrancar, avanzar, parar, ...

Atributos: color, velocidad, etc.

————→ Nombre de la clase  
————→ Métodos (funciones)  
————→ Atributos (datos)

Un **objeto** es una instancia de una clase

- Un objeto se distingue de otros miembros de la clase por sus atributos

Objeto Ferrari

Perteneiente a la  
clase coche



Nombre: Ferrari  
Métodos: arrancar, avanzar, parar, ...  
Atributos: color = "rojo";  
velocidad 300Km/h

- Una **clase** se declara, un **objeto** además se crea

# Repaso de Clases y Objetos

Para utilizar un objeto debemos instanciarlo. Se llama instanciar a la acción de crear un nuevo objeto dándole valores iniciales a nuestra clase.




**Nota:** Recuerda que para fines prácticos todo lo hicimos HARDCODEADO

# EL MENTANDO THIS

## THIS

En algunos lenguajes, dentro de la programación orientada a objetos, la palabra clave 'this' se utiliza para referenciar las propiedades propias del objeto, pertenecientes a la instancia actual, que estamos utilizando.

encontranos en [teloexplicocongatitos.com](https://teloexplicocongatitos.com)



todo lo que toca  
el 'this' es tuyo

wiiiiii

The logo consists of the text 'DEV.F.' in a bold, white, sans-serif font. The 'F' is stylized with a grid of small squares at its top right corner. The logo is centered within a dark blue diamond shape.

**DEV.F.**

## Pilares de la POO

*Programación Orientada a  
Objetos*



# Abstracción

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

# ENCAPSULAMIENTO



variable  
variable  
variable



## 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 si mismo a menos que sea necesario.



# 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 permita cambiar el color del cascabel del gato se podría acceder y cambiar esta información sin tocar otros atributos como el peso o la edad





# Herencia

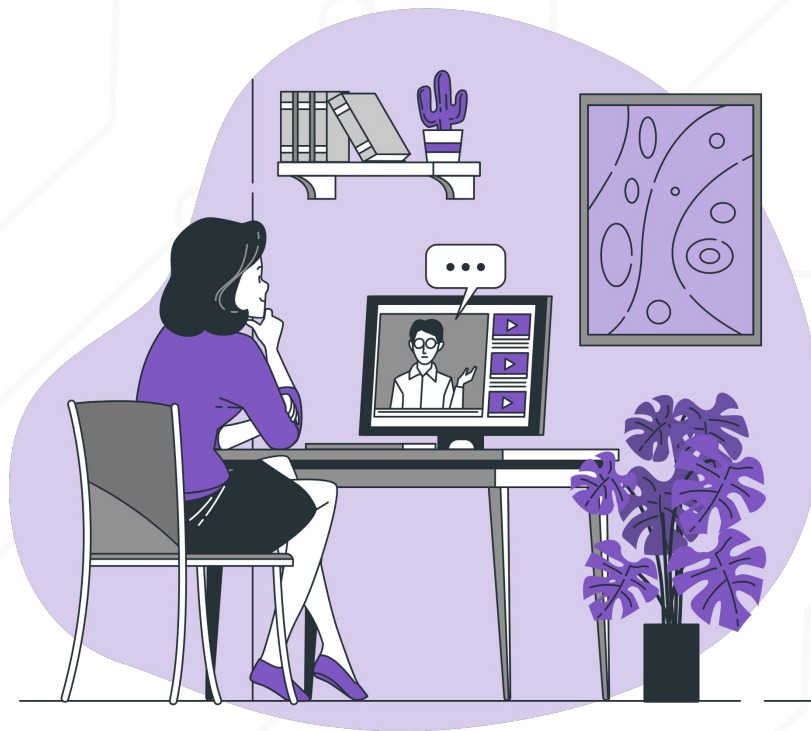
**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

# 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

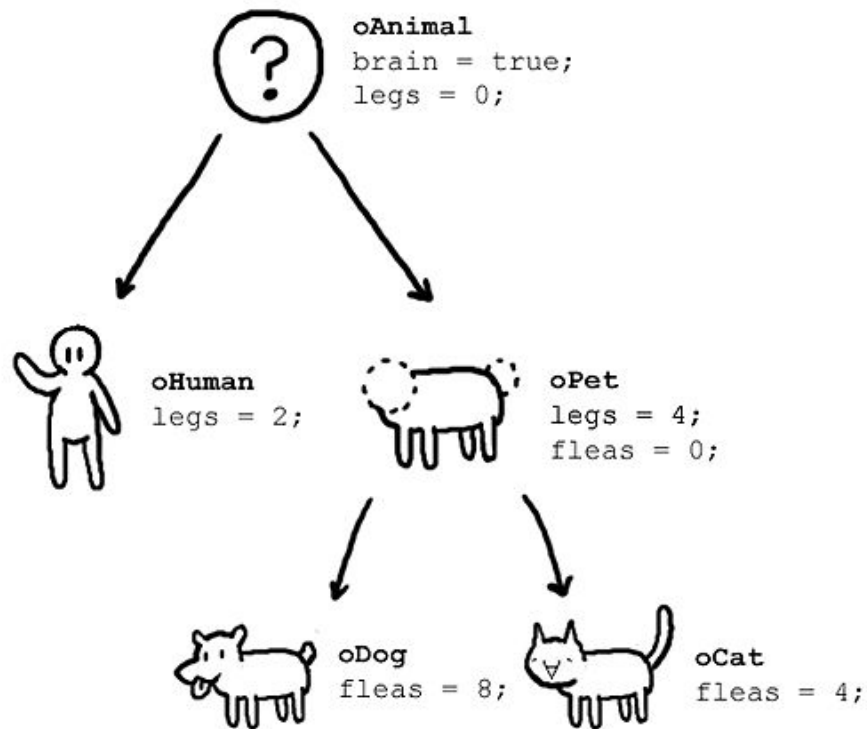
CLASE --> PADRE O SUPERCLASE

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

- Se crea una clase y se utiliza la palabra reservada **extends** (una clase que se crea utilizando herencia lo que hace es heredar todos los métodos de la clase padre o superclase)
- Crear una clase a partir de una existente.
- Superclase.
- Subclase.
- Se heredan atributos y métodos.

## 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".





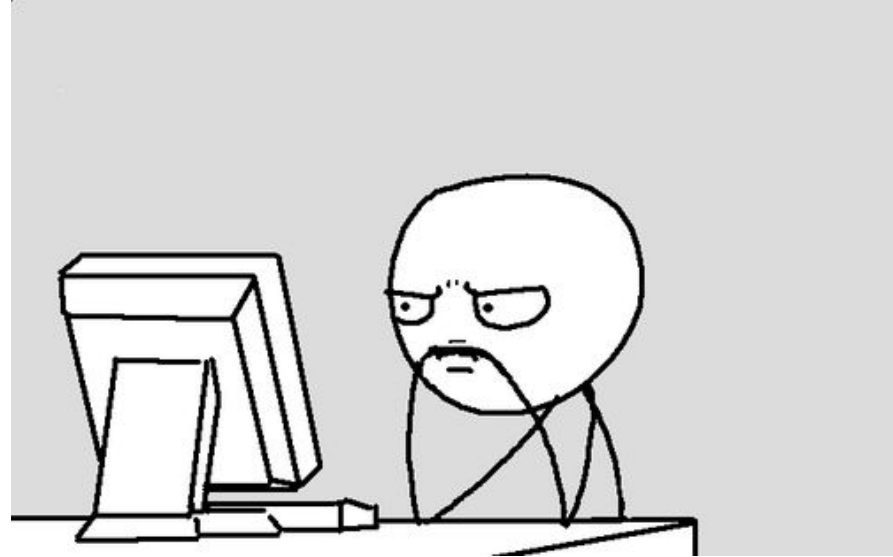
Si quisiera reproducir una película tendría los siguiente:

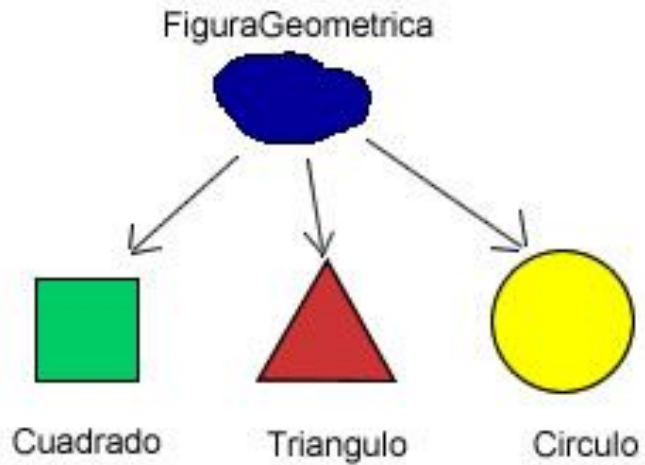
***reproducirPelicula( peliculaParaReproducir){... }***

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

***reproducirDocumental( documentaParaReproducir){... }***

**¿Realmente es  
necesario hacer dos  
métodos?**





# Polimorfismo

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