Programación orientada a objetos (POO)

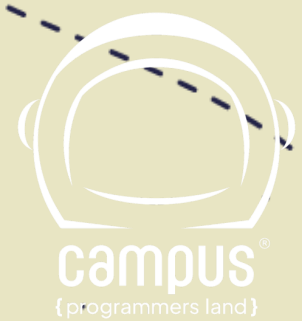
La programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos". Los objetos son entidades que representan conceptos del mundo real y que pueden tener propiedades (atributos) y realizar acciones (métodos).

En la programación orientada a objetos, los objetos son la base fundamental y se crean a partir de clases. Una clase es una plantilla o un molde que define las propiedades y comportamientos que tendrán los objetos que se creen a partir de ella.

Los principales conceptos de la programación orientada a objetos son:

Clase: Es una plantilla o definición que describe las características y comportamientos de los objetos que se pueden crear a partir de ella.

Objeto: Es una instancia de una clase. Representa un individuo o entidad específica y tiene sus propias propiedades y comportamientos.



Atributos: Son las propiedades o características de un objeto. Definen el estado de un objeto y se representan mediante variables en la clase.

Métodos: Son las acciones o comportamientos que un objeto puede realizar. Representan las operaciones que pueden realizarse con un objeto y se definen como funciones en la clase.

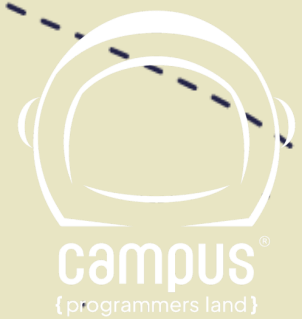
Encapsulación: Es el principio que establece que los atributos y métodos relacionados deben agruparse en una clase para ocultar los detalles internos y exponer solo una interfaz pública. Esto se logra mediante la especificación de niveles de acceso (público, privado, protegido) para los atributos y métodos.

Herencia: Es un mecanismo que permite crear nuevas clases basadas en clases existentes. La clase que se utiliza como base se denomina "clase padre" o "superclase", y la clase que se deriva se llama "clase hija" o "subclase". La herencia permite la reutilización de código y la creación de jerarquías de clases.

Nota: En JavaScript solo existen propiedades **publicas** y **privadas**

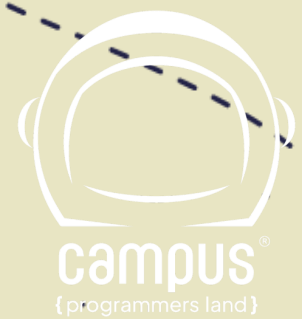
Polimorfismo: Es la capacidad de un objeto de tomar diferentes formas o comportarse de diferentes maneras según el contexto. Permite utilizar una interfaz común para objetos de diferentes clases y proporciona flexibilidad y extensibilidad en el diseño de programas.

La programación orientada a objetos permite organizar y modularizar el código de manera más clara y estructurada, lo que facilita el desarrollo, la mantenibilidad y la reutilización del código. Es ampliamente utilizado en diversos lenguajes de programación, como Java, C++, Python, PHP, entre otros.



Modificadores de acceso en JavaScript: Los modificadores de acceso son palabras clave utilizadas en la programación orientada a objetos para controlar la visibilidad y el acceso a los miembros (atributos y métodos) de una clase. Estos modificadores permiten establecer qué partes del código pueden acceder y modificar dichos miembros.

En JavaScript, hay dos modificadores de acceso para controlar la visibilidad de propiedades y métodos en una clase:



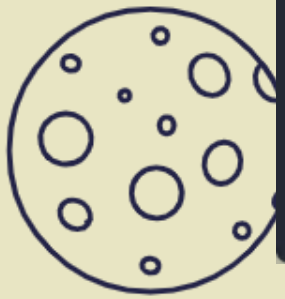
public: Los miembros declarados como public son accesibles desde cualquier lugar, ya sea desde dentro de la clase, desde las clases heredadas o desde fuera de la clase. Son visibles para todos.

private: Los miembros declarados como private solo son accesibles desde dentro de la misma clase en la que se definen. No pueden ser accedidos desde fuera de la clase, ni siquiera por las clases heredadas.

Nota: Es importante elegir adecuadamente los modificadores de acceso en función de las necesidades de diseño y la seguridad de la aplicación. Un buen diseño de clases utiliza el encapsulamiento y establece un acceso controlado a los miembros, evitando el acceso directo a los datos internos desde fuera de la clase y fomentando el uso de métodos para su manipulación.

Clases: Como vemos en el apartado introductorio a la programación orientada a objetos una clase es una plantilla que nos permite definir las características y comportamientos de los objetos que se pueden crear a partir de dicha clase a continuación veremos un ejercicio en el cual se evidencia como crear una clase y como instancia de una clase en JavaScript.

```
class Persona{
    ojos;
    #nombre;
    constructor(ojos, nombre){
        this.ojos = ojos;
        this.#nombre = nombre;
    }
    set setOjos(ojos){
        this.ojos = ojos;
    }
    get getOjos(){
        return this.ojos;
    }
    set setNombre(nombre){
        this.#nombre = nombre;
    }
    get getNombre(){
        return this.#nombre;
    }
    saludar(){
        return `Hola, mi nombre es ${this.#nombre} y mi color de ojos es ${this.ojos}`;
    }
}
```



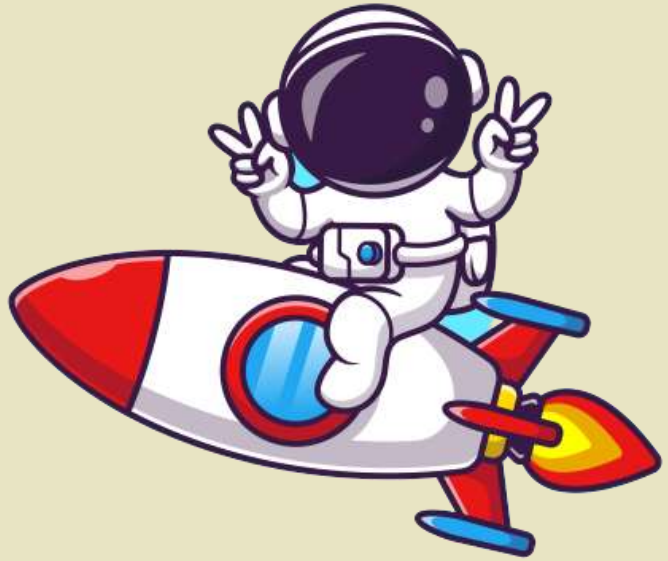
Instanciar clases: Se le llama instanciar una clase, crear un objeto o crear una instancia a la acción de crear un nuevo objeto basado en una clase particular. Esta acción la realizamos a través de la palabra clave **new**, seguida del nombre de la **clase**, la cuál puede tener parámetros, en cuyo caso se controlarían desde un **constructor**, concepto que veremos más a continuación.

```
let alumno = new Persona("Cafe", "Miguel");
```

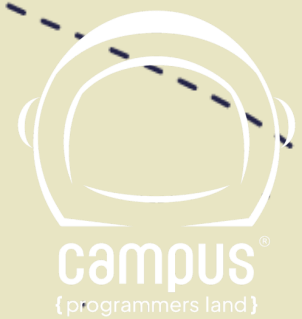
Acceder a la información de la instancia de la clase

```html
<body>
    <h1>Revisar la consola</h1>
    <div id="resultado">

    </div>
</body>
```

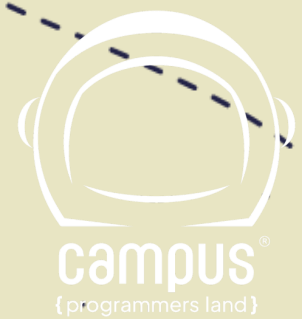


```
document.querySelector("#resultado").innerHTML = /*html*/`
    <h1>Nombre: <span>${alumno.getNombre}</span></h1>
    <h1>Ojos: <span>${alumno.getOjos}</span></h1>
`;
```



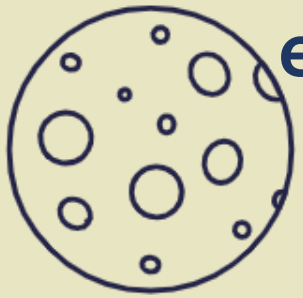
Métodos estáticos: En programación, un método estático es un método que pertenece a la clase en sí y no a una instancia específica de la clase. A diferencia de los métodos de instancia, los métodos estáticos se pueden llamar directamente en la clase sin necesidad de crear un objeto o instancia de la misma.

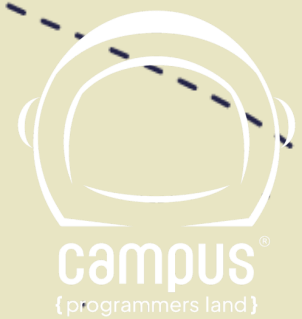
Algunas características importantes de los métodos estáticos en JavaScript son:



No requieren una instancia: Los métodos estáticos se pueden invocar directamente desde la clase, utilizando la sintaxis `Clase.metodoEstatico()`, sin necesidad de crear un objeto de la clase.

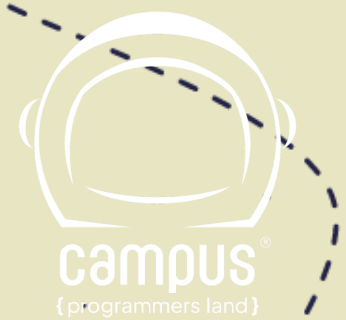
No pueden acceder a propiedades de instancia: Los métodos estáticos no pueden acceder directamente a las propiedades de instancia de la clase, ya que no tienen una instancia específica asociada. Solo pueden acceder a propiedades estáticas (variables estáticas) que pertenezcan a la clase.





No pueden utilizar this: En un método estático, no se puede utilizar la palabra clave **this** para hacer referencia a la instancia actual de la clase, ya que no hay una instancia asociada.

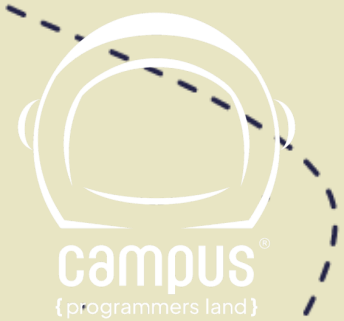
Útiles para utilidades compartidas: Los métodos estáticos son útiles para definir funciones o utilidades que no dependen del estado de una instancia específica. Se pueden utilizar para operaciones globales, cálculos matemáticos, acceso a bases de datos, manipulación de archivos, etc.



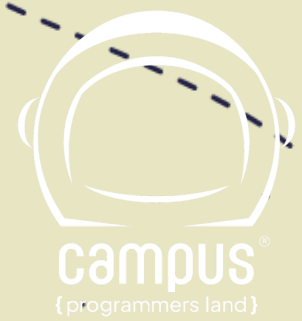
```
class Persona{
    ojos;
    #nombre;
    static nombreAux;
    constructor(ojos, nombre){
        this.ojos = ojos;
        this.#nombre = nombre;
    }
    set setOjos(ojos){
        this.ojos = ojos;
    }
    get getOjos(){
        return this.ojos;
    }
    set setNombre(nombre){
        this.#nombre = nombre;
    }
    get getNombre(){
        return this.#nombre;
    }
    static saludarAux(){
        super.nombreAux = "xxxxxx";
        return `Hola como estas ${Persona.nombreAux}`;
    }
}
document.querySelector("#resultado").innerHTML = /*html*/`
    <h1>Nombre: <span>${Persona.saludarAux()}</span></h1>
`;
```

Nota: Palabra clave this: En las clases de JavaScript, la palabra "this" se usa para referirse al objeto en el que te encuentras actualmente. Te permite acceder a las cosas (propiedades y métodos) que pertenecen a esa clase y no a otras partes del código. En resumen, "this" se usa dentro de una clase para hablar sobre sí misma y lo que le pertenece.

La palabra clave super: En JavaScript, la palabra reservada "super" se utiliza para llamar y acceder a los métodos y propiedades de la clase padre, dentro de una clase hija. Proporciona una forma de extender y heredar el comportamiento de la clase padre en la clase hija.



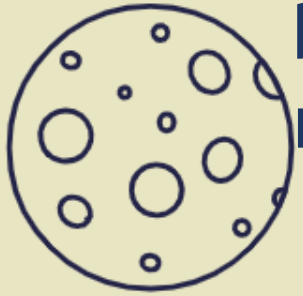
Herencia: La herencia en programación es un concepto que permite crear nuevas clases basadas en clases existentes, aprovechando y extendiendo su funcionalidad. La clase existente se conoce como clase base o clase padre, mientras que la nueva clase creada se llama clase derivada o clase hija.

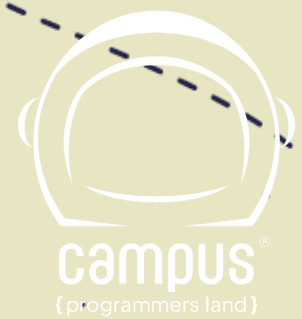


Algunos conceptos importantes relacionados con la herencia son los siguientes:

Clase base / Clase padre: Es la clase original de la cual se deriva una nueva clase. Define los atributos y métodos básicos que serán heredados por las clases derivadas.

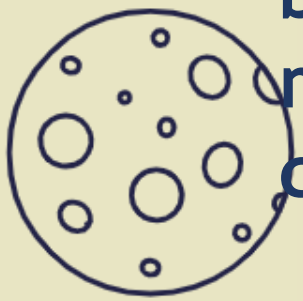
Clase derivada / Clase hija: Es la nueva clase creada que se basa en la clase base. Hereda los atributos y métodos de la clase base y puede agregar nuevos atributos y métodos, así como modificar o ampliar los existentes.





Herencia simple y herencia múltiple: La herencia simple se refiere a la relación en la que una clase derivada hereda de una sola clase base. Por otro lado, la herencia múltiple se refiere a la relación en la que una clase derivada hereda de múltiples clases base. No todos los lenguajes de programación admiten la herencia múltiple.

Polimorfismo: El polimorfismo es la capacidad de un objeto de una clase derivada para ser tratado como un objeto de su clase base. Esto permite utilizar una referencia de la clase base para manipular objetos de diferentes clases derivadas sin tener que conocer la clase concreta en tiempo de compilación.



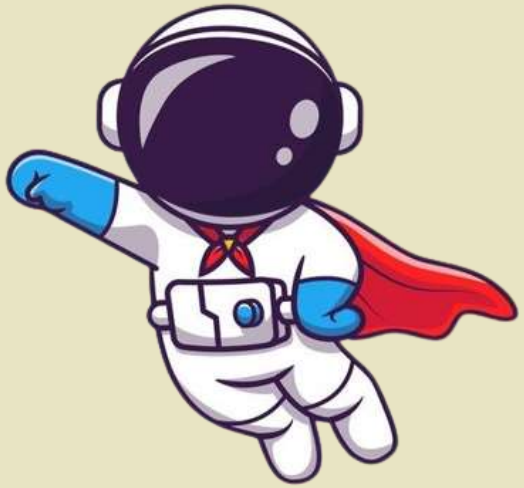
Clase padre

```
class Transporte {
    #rueda;
    #capacidad;
    constructor(rueda, capacidad) {
        this.#rueda = rueda;
        this.#capacidad = capacidad;
    }
    get getInfo() {
        return `El transporte tiene ${this.#rueda} ruedas y una capacidad de ${this.#capacidad} personas`;
    }
    get getRuedas() {
        return this.#rueda;
    }
    get getCapacidad() {
        return this.#capacidad;
    }
}
```

Clases Hijas

```
class Bicicleta extends Transporte {
    constructor(rueda, capacidad) {
        super(rueda, capacidad);
    }
    get getInfo() {
        return `El transporte tiene ${this.getRuedas} ruedas y una capacidad de ${this.getCapacidad} personas y NO GASTA GASOLINA`;
    }
}
```

```
class Automovil extends Transporte{
    transmicion;
    constructor(rueda, capacidad, transmicion) {
        super(rueda, capacidad);
        this.transmicion = transmicion;
    }
    get getTransmicion() {
        return this.transmicion;
    }
}
```



campus
{programmers land}

Nota: para acceder a las clases hay que instanciar las calses por separado para ver la magia de la herencia.

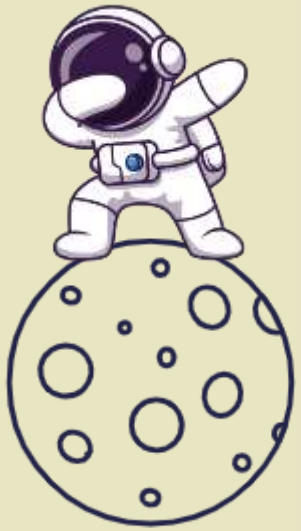
```
let res = document.querySelector("#resultado");
let bicicleta = new Bicicleta(2, 1);
res.insertAdjacentHTML("beforeend", `<h1>${bicicleta.getInfo}</h1>`);
let automovil = new Automovil(4, 4, "Manual");
res.insertAdjacentHTML("beforeend", `<h1>${automovil.getInfo} tranmision: ${automovil.getTransmicion}</h1>`);
```

¡Felicidades por haber aprendido Programación Orientada a Objetos (POO) con JavaScript! Este es un logro significativo y demuestra tu dedicación y esfuerzo en el aprendizaje de esta poderosa forma de programación.

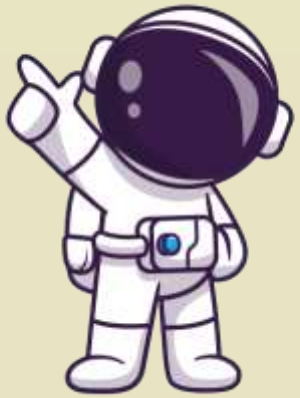
La POO es una herramienta invaluable que te permitirá organizar y estructurar tu código de manera eficiente, permitiéndote crear programas más claros, modulares y reutilizables. Con los conceptos que has dominado, ahora puedes diseñar y construir sistemas más complejos y robustos.

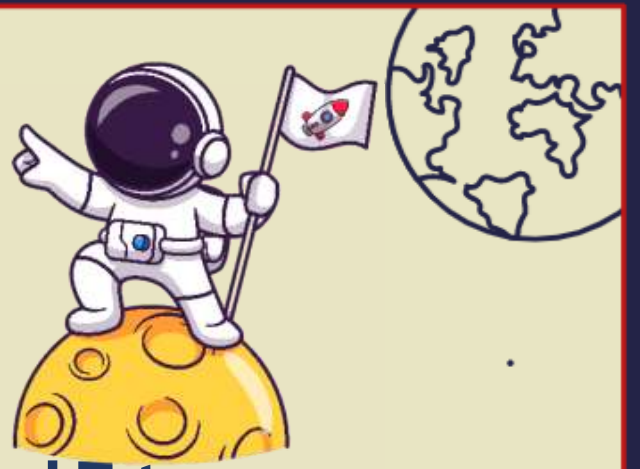
No olvides que el aprendizaje es un viaje constante, y la POO es solo una de las muchas habilidades valiosas en el mundo de la programación. Continúa explorando, practicando y desafiándote a ti mismo en este apasionante campo.

Recuerda que cada nuevo conocimiento que adquieres te brinda más herramientas para desarrollar soluciones creativas a los desafíos que te encuentres. Nunca dudes de tu capacidad para aprender y mejorar continuamente.



campus
{programmers land}





¡Sigue adelante con tu entusiasmo y determinación! Estoy seguro de que tu dominio de la POO en JavaScript te abrirá puertas a emocionantes oportunidades y te permitirá construir proyectos increíbles. ¡El futuro está lleno de posibilidades y tú estás preparado para aprovecharlas al máximo!

¡Mucho éxito en tu camino como programador/a orientado/a a objetos con JavaScript y en todas tus futuras aventuras en el mundo de la programación! ¡Continúa aprendiendo, creciendo y desafiándote a ti mismo/a!