🏗️ Programación Orientada a Objetos (POO) en JavaScript

La POO es un paradigma de programación que organiza el software alrededor de **objetos**, que son instancias de **clases**. En esencia, permite modelar entidades del mundo real en tu código, agrupando datos (propiedades) y el comportamiento asociado a esos datos (métodos).

1. El Concepto de Clase (class)

Una **clase** es, fundamentalmente, una **plantilla** o un *plano* para crear objetos. Define la estructura de datos que tendrá el objeto (propiedades) y las funciones que podrá realizar (métodos).

Aunque históricamente JavaScript se basó en prototipos, las palabras clave class (introducidas en ES6) son la forma moderna y la más clara de implementar la POO, ofreciendo lo que se conoce como "azúcar sintáctica" sobre el modelo de prototipos subyacente.

Ejemplo de Clase Simple:

JavaScript

```
class Automovil {
  // El constructor se ejecuta al crear una nueva instancia (objeto)
```

```
  constructor(marca, modelo) {
    this.marca = marca; // Propiedad
    this.modelo = modelo; // Propiedad
    this.encendido = false; // Propiedad inicial
  }

  // Método (comportamiento)
  encender() {
    this.encendido = true;
    return `El ${this.marca} ${this.modelo} se ha encendido.`;
  }

  // Otro Método
  estado() {
    return this.encendido ? 'El motor está en marcha.' : 'El motor está apagado.';
  }
}

// Creación de una instancia (Objeto)
const miCoche = new Automovil('Toyota', 'Corolla');

console.log(miCoche.estado()); // Salida: El motor está apagado.
console.log(miCoche.encender()); // Salida: El Toyota Corolla se ha encendido.
console.log(miCoche.estado()); // Salida: El motor está en marcha.
```

🌟 Características Clave de la POO

La POO se define por cuatro pilares principales, todos soportados en JavaScript:

1. Encapsulamiento (Encapsulation)

- **Definición:** Consiste en agrupar los **datos** (propiedades) y los **métodos** que operan sobre esos datos dentro de la unidad de la clase. Además, implica restringir el acceso directo a algunas de las propiedades del objeto.
- **En JavaScript:** Históricamente, JavaScript no tenía modificadores de acceso (public, private). Sin embargo, el **encapsulamiento real** se logra ahora usando el prefijo # (hash) para declarar **campos privados** dentro de una clase.

Ejemplo:

JavaScript

```
class CuentaBancaria {
  #saldo = 0; // Campo privado: solo accesible dentro de la clase

  constructor(montoInicial) {
    this.#saldo = montoInicial;
  }

  depositar(monto) {
    if (monto > 0) {
      this.#saldo += monto;
    }
  }
```

```
  // Getter (método público para acceder al dato privado)
  verSaldo() {
    return `Saldo actual: $${this.#saldo}`;
  }
}

const cuenta = new CuentaBancaria(500);
// console.log(cuenta.#saldo); // ¡Esto causaría un error! (Acceso privado)
console.log(cuenta.verSaldo()); // Salida: Saldo actual: $500
```

2. Herencia (Inheritance)

- **Definición:** Permite que una nueva clase (clase hija o subclase) herede las propiedades y métodos de una clase existente (clase padre o superclase). Esto promueve la **reutilización de código**.
- **En JavaScript:** Se utiliza la palabra clave extends. Para ejecutar el constructor de la clase padre desde la subclase, se usa super().

Ejemplo:

JavaScript

```
class Bicicleta extends Automovil { // Bicicleta hereda de Automovil
  constructor(marca, modelo, tipo) {
    super(marca, modelo); // Llama al constructor de Automovil
```

```
    this.tipo = tipo; // Propiedad específica de Bicicleta
  }

  // Método específico (sobrescribe el método 'estado' de la clase padre)
  estado() {
    return `La bicicleta ${this.marca} es de tipo ${this.tipo}.`;
  }
}

const miBici = new Bicicleta('Trek', 'FX', 'Híbrida');

// miBici tiene acceso al método estado, pero usa su propia implementación.
console.log(miBici.estado()); // Salida: La bicicleta Trek es de tipo Híbrida.
```

3. Polimorfismo (Polymorphism)

- **Definición:** Significa "muchas formas". Permite que objetos de diferentes clases respondan al **mismo mensaje** (nombre de método) de diferentes maneras.
- **En JavaScript:** Se ve claramente en la **sobreescritura de métodos** (*method overriding*), como en el ejemplo de estado() anterior. La clase Automovil y Bicicleta tienen un método llamado estado(), pero la lógica de cada uno es diferente.

4. Abstracción (Abstraction)

- **Definición:** Es el proceso de mostrar solo la información esencial y ocultar los detalles de implementación complejos. Se centra en *qué hace* el objeto, no en *cómo lo hace*.

- **En JavaScript:** Se logra diseñando interfaces claras (los métodos públicos de la clase) y ocultando la complejidad interna (como el uso de la función super() y los campos privados # que manejan los detalles internos del objeto). Por ejemplo, cuando llamas a miCoche.encender(), no necesitas saber exactamente cómo está implementada internamente la lógica de encendido, solo que el coche se enciende.