💡 ¿Qué es Encapsulación?

La **encapsulación** es uno de los principios fundamentales de la Programación Orientada a Objetos (POO). Permite agrupar datos (atributos) y métodos (funciones) dentro de una clase, y controlar quién tiene acceso a esos datos.

🔐 Modificadores de Acceso

En JavaScript, los atributos y métodos pueden tener distintos niveles de acceso:

Modificador	Descripción	Ejemplo
Public	Atributos y métodos accesibles desde cualquier parte del código.	`this.nombre = "Gamal";`
Private	Solo accesibles desde dentro de la clase. Se declaran con `#`.	`#saldo = 1000;`

✍️ Ejemplo práctico

js

```
class CuentaBancaria {
  // Atributo privado
  #saldo = 0;

  constructor(titular) {
    this.titular = titular;
  }

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

  retirar(monto) {
    if (monto > 0 && monto <= this.#saldo) {
      this.#saldo -= monto;
    }
  }
```

```
  // Getter para acceder al saldo
  getSaldo() {
    return this.#saldo;
  }
}

const cuenta1 = new CuentaBancaria("Gamal");
cuenta1.depositar(1000);
cuenta1.retirar(300);
console.log(cuenta1.getSaldo()); // Muestra 700
```

🔗 **Referencias:**

- MDN - Clases en JavaScript
- MDN - Campos Privados (Private Fields)

🤓 Preguntas orientadoras

1. ¿Qué ventajas tiene encapsular datos dentro de una clase?
2. ¿Por qué es importante limitar el acceso a ciertos atributos?
3. ¿Qué sucede si intentás acceder a un atributo privado desde fuera de la clase?
4. ¿Podemos modificar un atributo privado sin un método público?
5. ¿Qué pasa si se intenta modificar un atributo privado sin usar un `setter`?

💡 Validaciones en métodos

Las validaciones nos permiten controlar que los datos sean correctos antes de realizar operaciones. Por ejemplo, en un método de "depósito", podemos validar que el monto sea positivo.

✍️ Ejemplo práctico

js

```
class Producto {
  #precio = 0;

  constructor(nombre, precio) {
    this.nombre = nombre;
    this.setPrecio(precio);
  }

  // Método para modificar el precio
  setPrecio(precio) {
    if (precio > 0) {
      this.#precio = precio;
    } else {
      console.error("El precio no puede ser negativo");
    }
  }

  getPrecio() {
    return this.#precio;
  }
}

const producto1 = new Producto("Monitor", 1500);
console.log(producto1.getPrecio()); // 1500
producto1.setPrecio(-500);          // Error: El precio no puede ser
negativo
```

🔗 **Referencias:**

- MDN - Encapsulación en JavaScript
- MDN - Métodos Getter y Setter
