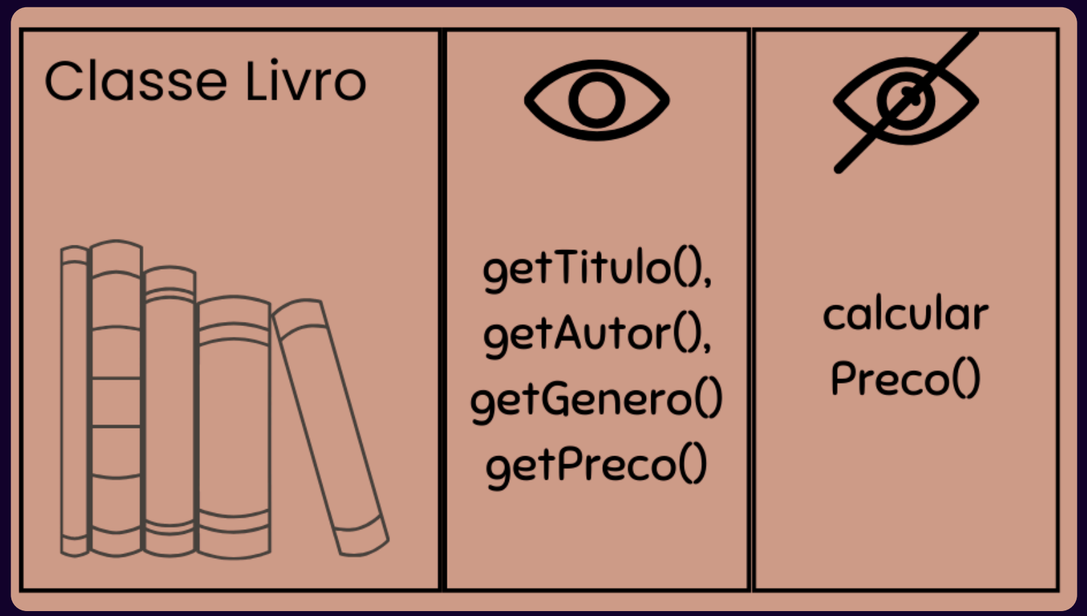
ENCAPSULAMENTO

O QUE É ENCAPSULAMENTO?

Encapsulamento é o **princípio** da *programação orientada a objetos* que consiste em **esconder** os detalhes internos de um objeto e permitir acesso apenas ao que for **necessário**.

Em outras palavras, é como colocar os **dados** e **comportamentos** "dentro de uma caixinha" (objeto) e **limitar** o que pode ser acessado de fora.



POR QUE USAR ENCAPSULAMENTO?

01 **Protege os dados** de serem alterados de forma incorreta.

02 **Evita efeitos colaterais** inesperados no código.

03 **Facilita manutenção** e evolução do código.

04 **Define contratos de uso** (o que pode ou não ser acessado).

COMO FAZER ENCAPSULAMENTO EM JAVASCRIPT?

1. Usando convenção (prefixo _)

```
class ContaBancaria {
  constructor(saldoInicial) {
    this._saldo = saldoInicial; // o "_" indica "privado" por convenção
  }

  depositar(valor) {
    this._saldo += valor;
  }

  verSaldo() {
    return this._saldo;
  }
}

const conta = new ContaBancaria(100);
conta.depositar(50);
console.log(conta.verSaldo()); // 150
console.log(conta._saldo); // Ainda acessível! ⚠️
```

COMO FAZER ENCAPSULAMENTO EM JAVASCRIPT?

2. Usando # para campos privados (ES2022+)

```
1  class ContaBancaria {
2    #saldo; // campo realmente privado
3
4    constructor(saldoInicial) {
5      this.#saldo = saldoInicial;
6    }
7
8    depositar(valor) {
9      this.#saldo += valor;
10   }
11
12   verSaldo() {
13     return this.#saldo;
14   }
15 }
16
17 const conta = new ContaBancaria(100);
18 conta.depositar(50);
19 console.log(conta.verSaldo()); // 150
20 console.log(conta.#saldo); // ❌ Erro! Campo privado
```

CONCLUSÃO

Pense em um cofre: você pode colocar e tirar dinheiro com a chave (método), mas não pode enfiar a mão e mudar o saldo diretamente. Isso é encapsulamento.

MÉTODOS ESPECIAIS

MÉTODOS GETTERS E SETTERS?

São **métodos especiais** usados para acessar ou modificar **valores privados** (ou controlados) de um objeto. Eles servem como “portas” para ler ou alterar um dado, aplicando regras se necessário.

POR QUE USAR?

01 Encapsular acesso a dados.

02 Validar valores antes de alterar.

03 Simular atributos como se fossem públicos, mas mantendo controle.

04 Criar APIs de objetos mais intuitivas.

COMO FUNCIONAM?

```
class Pessoa {
  constructor(nome) {
    this._nome = nome;
  }

  get nome() {
    return this._nome;
  }

  set nome(novoNome) {
    if (typeof novoNome === 'string') {
      this._nome = novoNome;
    } else {
      console.error('Nome precisa ser uma string');
    }
  }
}

const p = new Pessoa('João');
console.log(p.nome); // acessa como se fosse uma propriedade
p.nome = 'Maria';    // altera, mas passa pelo set
console.log(p.nome); // 'Maria'

```

DETALHES IMPORTANTES:

- O *get* **não precisa de parênteses** na chamada: *obj.propriedade*.
- O *set* é chamado automaticamente ao atribuir: *obj.propriedade* = *valor*.
- Mesmo sendo funções, se comportam como propriedades comuns.

Quando usar?

Use quando quiser:

- Validar ou transformar dados antes de armazenar.
- Evitar acesso direto a propriedades sensíveis.
- Criar APIs mais limpas, tipo: obj.nome = 'Lucas' em vez de obj.setNome('Lucas').

SAIBA MAIS

GET

SET

STATIC?

Em JavaScript, o *static* define **métodos ou propriedades que pertencem à classe, não** às instâncias (objetos criados com *new*).

MÉTODO STATIC?

Ou seja: você chama direto pela **classe**, e **não** por um objeto dela.

QUANDO USAR?

Use quando o método:

- **Não depende de dados do objeto.**
- Serve como **utilitário** ou **função auxiliar**.
- Precisa ser compartilhado entre todas as instâncias sem duplicar código.

EXEMPLO PRÁTICO:

```
1  class Conversor {
2    static celsiusParaFahrenheit(celsius) {
3      return (celsius * 9/5) + 32;
4    }
5
6    static fahrenheitParaCelsius(fahrenheit) {
7      return (fahrenheit - 32) * 5/9;
8    }
9  }
10
11 console.log(Conversor.celsiusParaFahrenheit(30)); // 86
12 console.log(Conversor.fahrenheitParaCelsius(86)); // 30
```

CONCLUSÃO

Regras:

- Métodos *static* **não acessam** *this* **da instância**, apenas da **classe**.
- Você **não pode chamar um** *static* **com um objeto**, apenas com a classe.

Uso comum:

- Métodos utilitários (ex: formatar datas, gerar ID, converter unidades)
- Factories (Pessoa.criarAnonimo())
- Operações matemáticas (como *Math.sqrt()* que também é *static*)

SAIBA MAIS

STATIC

THANK YOU

@wallace027dev