

# Encapsulamento

DEVinHouse

Parcerias para desenvolver a sua carreira

**SENAI**

<LAB365>

# AGENDA

- Revisão
- Encapsulamento
- Closures

# Criando a nossa classe Carro

```
1  class Carro {  
2      constructor(modelo, velocidade) {  
3          this.modelo = modelo;  
4          this.velocidade = velocidade;  
5      }  
6  
7      acelerar() {  
8          /* código do carro para acelerar */  
9      }  
10  
11     frear() {  
12         /* código do carro para frear */  
13     }  
14  
15     acenderFarol() {  
16         /* código do carro para acender o farol */  
17     }  
18 }
```

Voltando a nossa analogia do carro, sabemos que ele possui atributos e métodos, ou seja, características e comportamentos.

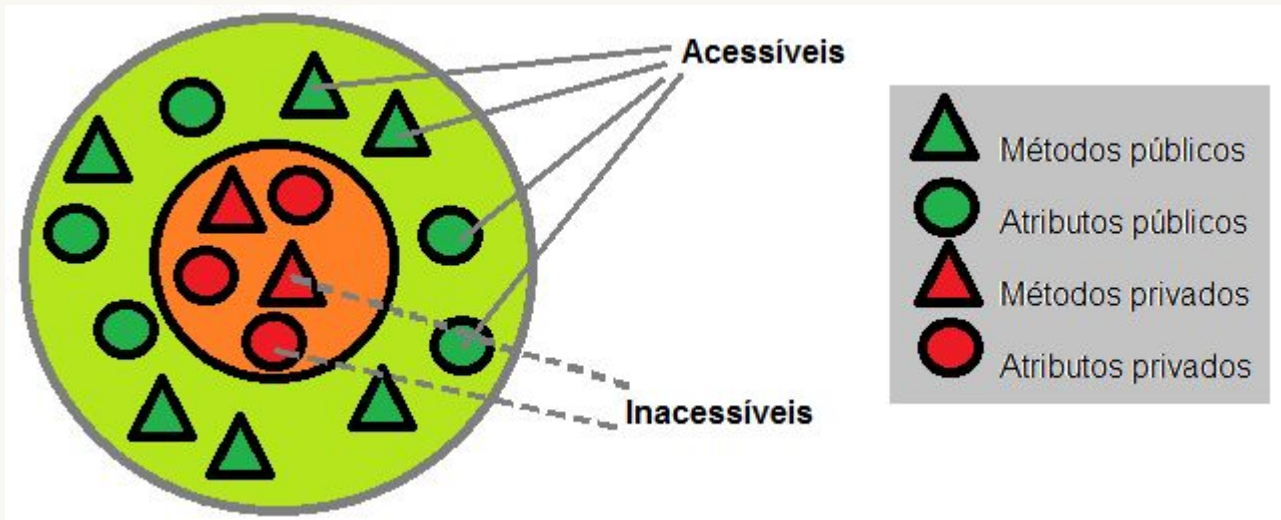
Os métodos do carro, como acelerar, podem usar atributos e outros métodos do carro como o tanque de gasolina e o mecanismo de injeção de combustível, respectivamente, uma vez que acelerar gasta combustível.

No entanto, se alguns desses atributos ou métodos forem facilmente visíveis e modificáveis, como o mecanismo de aceleração do carro, isso pode dar liberdade para que alterações sejam feitas, resultando em efeitos colaterais imprevisíveis.

Nessa analogia, uma pessoa pode não estar satisfeita com a aceleração do carro e modifica a forma como ela ocorre, criando efeitos colaterais que podem fazer o carro nem andar, por exemplo.

Dizemos, nesse caso, que o método de aceleração do seu carro não é visível por fora do próprio carro. Na POO, um atributo ou método que não é visível de fora do próprio objeto é chamado de "privado" e quando é visível, é chamado de "público".

# Encapsulamento



Mas então, como sabemos como o nosso carro acelera? É simples: não sabemos. Nós só sabemos que para acelerar, devemos pisar no acelerador e de resto o objeto sabe como executar essa ação sem expor como o faz.

Dizemos que a aceleração do carro está encapsulada, pois sabemos o que ele vai fazer ao executarmos esse método, mas não sabemos como.



O mesmo vale para atributos. Por exemplo: não sabemos como o carro sabe qual velocidade mostrar no velocímetro ou como ele calcula sua velocidade, mas não precisamos saber como isso é feito.

Só precisamos saber que ele vai nos dar a velocidade certa.

É para isso que só criamos nossas variáveis(let ou const) dentro do escopo que nós queremos utilizá-las.

Para ligar o carro, você precisa apenas virar uma chave ou apertar um botão. Não é necessário conectar fios debaixo do capô, girar o virabrequim e cilindros, e iniciar o ciclo de funcionamento do motor.

Os detalhes estão escondidos debaixo do capô do carro. Você tem apenas uma interface simples: uma chave para dar a partida, um volante e alguns pedais.

Esta ilustração mostra como cada objeto tem uma interface: Uma parte pública de um objeto, aberta para interações com outros objetos.

# Encapsulamento

Encapsulamento é a capacidade do objeto de ocultar de outros objetos partes de seu estado e comportamentos, expondo ao resto do programa apenas uma interface limitada.

# Encapsulamento

Encapsular algo significa torná-lo privado, e portanto acessível apenas dentro dos métodos de sua própria classe.

## Benefícios do Encapsulamento

- Protege a integridade do objeto ao impedir que usuários alterem dados internos do componente para valores inválidos ou inconsistentes.
- Se algum código específico mudar a propriedade `carrinho.valorTotal` sem que se tenha adicionado ou removido um item do carrinho, o valor estará incorreto.
- Reduz a complexidade do sistema, permitindo que o desenvolvedor limite as interdependências entre os componentes do software.
- `comanda.calcularTotal()` é mais simples do que ter que pegar todas as propriedades, fazer os calculos e adicionar o valor extra do garçom toda vez que o sistema for calcular o valor total do jantar.

Mão na Massa!

# Encapsulamento

```
class Pessoa {  
  nome  
  idade  
  
  constructor(nome, idade) {  
    this.nome = nome  
    this.idade = idade  
  }  
  
  exibir() {  
    return `O meu nome é ${this.nome} e tenho ${this.idade} anos.`  
  }  
}  
  
const p1 = new Pessoa("Pedro", 21)  
console.log(p1.exibir())  
console.log(p1.nome)  
console.log(p1.idade)  
console.log(JSON.stringify(p1))  
console.log(Object.keys(p1))  
console.log(Object.values(p1))  
  
for(let atributos in p1) {  
  console.log(p1[atributos], atributos)  
}
```



# Encapsulamento

```
class Pessoa {
  #nome
  #idade

  constructor(nome, idade) {
    this.#nome = nome
    this.#idade = idade
  }

  exibir() {
    return `O meu nome é ${this.#nome} e tenho ${this.#idade} anos.`
  }
}

const p1 = new Pessoa("Pedro", 21)
console.log(p1.exibir())
console.log(p1.#nome)
console.log(p1.#idade)
```

node "c:\Users\Ana Paula Marianni\Desktop\testando\cd.js"  
c:\Users\Ana Paula Marianni\Desktop\testando\cd.js:17  
console.log(p1.#nome)  
 ^  
TypeError: Private field '#nome' must be declared in an enclosing class  
 at wrapSafe (internal/modules/cjs/loader.js:988:16)  
 at Module.\_compile (internal/modules/cjs/loader.js:1036:27)  
 at Object.Module.\_extensions..js (internal/modules/cjs/loader.js:1101:10)  
 at Module.load (internal/modules/cjs/loader.js:937:32)  
 at Function.Module.\_load (internal/modules/cjs/loader.js:778:12)  
 at Function.executeUserEntryPoint [as runMain] (internal/modules/run\_main.js:76:12)

# Encapsulamento

```
1  class Pessoa {
2      #nome
3      #idade
4
5      constructor(nome, idade) {
6          this.#nome = nome
7          this.#idade = idade
8      }
9
10     get nome(){
11         return this.#nome
12     }
13
14     get nome(){
15         return this.#idade
16     }
17
18     exibir() {
19         return `O meu nome é ${this.nome} e tenho ${this.
20             idade} anos.`
21     }
22
23     const p1 = new Pessoa("Pedro", 21)
24     console.log(p1.exibir())
25     console.log(p1.nome)
26     console.log(p1.idade)
```

PROBLEMS OUTPUT ...

Code

```
[Running] node "c:\Users\Ana Paula Marianni\Desktop\testando\c
O meu nome é 21 e tenho undefined anos.
21
undefined
```

# Encapsulamento

```
15     return this.#idade
16 }
17
18 set nome(nome) {
19     this.#nome = nome
20 }
21
22 exibir() {
23     return `O meu nome é ${this.nome} e tenho ${this.
24         idade} anos.`
25 }
26
27 const p1 = new Pessoa("Pedro", 21)
28 p1.nome = "Ana"
29 console.log(p1.nome)
30 console.log(p1.exibir())
31 console.log(p1.nome)
32 console.log(p1.idade)
33 console.log(JSON.stringify(p1))
34 console.log(Object.keys(p1))
35 console.log(Object.values(p1))
36
37 for(let atributos in p1) {
38     console.log(p1[atributos], atributos)
39 }
```

PROBLEMS OUTPUT ...

Code

```
[Running] node "c:\Users\Ana Paula Marianni\Desktop\testando\cd.js"
Ana
O meu nome é Ana e tenho 21 anos.
Ana
21
```

## Closures

Um closure é uma função que se "lembra" do ambiente — ou escopo léxico — em que ela foi criada.

Em outras palavras.. Closure é a forma de fazer com que as variáveis dentro de uma função sejam privadas e persistentes.

# Closures



```
function f() {  
  let valor = "Estou aqui!";  
  
  return function() {  
    alert(valor);  
  }  
}  
  
let g = f();
```

> console.dir(g)

VM274:1

```
▼ f anonymous() ⓘ  
  arguments: null  
  caller: null  
  length: 0  
  name: ""  
  ► prototype: {constructor: f}  
    [[FunctionLocation]]: VM268:4  
  ► [[Prototype]]: f ()  
  ▼ [[Scopes]]: Scopes[3]  
    ► 0: Closure (fu) {valor: 'Estou aqui!'}  
    ► 1: Script {LoadTimeData: f, g: f}  
    ► 2: Global {window: Window, self: Window, document: ...}
```

Classes - <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes>

Closures -

<https://medium.com/@stephanowallace/javascript-mas-afinal-o-que-s%C3%A3o-closures-4d67863ca9fc>

[Closures: Muitos Programadores NÃO conhecem esse CONCEITO! IMPORTANTE para JAVASCRIPT! - YouTube](#)

POO - <https://www.alura.com.br/artigos/poo-programacao-orientada-a-objetos>



# DEVinHouse

Parcerias para desenvolver a sua carreira

**OBRIGADO!**



<LAB365>