JavaScript POO - Guia Prático

Métodos em Objetos

Métodos são funções atribuídas como propriedades de um objeto.

Exemplo:

```
const pessoa = {
    nome: 'João',
    falar() {
        console.log(`Olá, meu nome é ${this.nome}`);
    }
};
pessoa.falar(); // Saída: Olá, meu nome é João
```

Prototypes

Prototypes permitem a herança entre objetos. Cada objeto em JavaScript tem um protótipo.

Exemplo:

```
function Pessoa(nome) {
    this.nome = nome;
}
Pessoa.prototype.falar = function() {
    console.log(`Meu nome é ${this.nome}`);
};
```

```
const joao = new Pessoa('João');

joao.falar(); // Saída: Meu nome é João
```

Classes

Classes são uma sintaxe mais clara para criar objetos e lidar com herança.

Exemplo:

```
class Pessoa {
    constructor(nome) {
        this.nome = nome;
    }
    falar() {
        console.log(`Meu nome é ${this.nome}`);
    }
}
const joao = new Pessoa('João');

joao.falar(); // Saída: Meu nome é João
```

Instância de Classe por Função

Podemos instanciar classes usando funções.

Exemplo:

```
function Carro(modelo) {
    this.modelo = modelo;
}
Carro.prototype.ligar = function() {
    console.log(`${this.modelo} está ligado.`);
};
const carro1 = new Carro('Fusca');
carro1.ligar(); // Saída: Fusca está ligado.
```

Instância de Classe por new

O operador `new` cria uma nova instância de um objeto baseado em uma função ou classe.

Exemplo:

```
class Animal {
    constructor(tipo) {
        this.tipo = tipo;
    }
}
const cachorro = new Animal('Cachorro');
console.log(cachorro.tipo); // Saída: Cachorro
```

Métodos Prototype

Os métodos adicionados ao prototype ficam disponíveis para todas as instâncias.

Exemplo:

```
function Produto(nome) {
    this.nome = nome;
}
Produto.prototype.mostrar = function() {
    console.log(`Produto: ${this.nome}`);
};
const p1 = new Produto('Caneta');
p1.mostrar(); // Saída: Produto: Caneta
```

Override no Prototype

É possível sobrescrever métodos no protótipo.

Exemplo:

```
Produto.prototype.mostrar = function() {
    console.log(`Item: ${this.nome}`);
};
p1.mostrar(); // Saída: Item: Caneta
```

Symbol

Symbols são valores únicos que podem ser usados como identificadores.

Exemplo:

```
const simbolo = Symbol('id');

const obj = {
    [simbolo]: 123
};

console.log(obj[simbolo]); // Saída: 123
```

Getters e Setters

Getters e Setters permitem controlar o acesso às propriedades.

Exemplo:

```
class Pessoa {
    constructor(nome) {
        this._nome = nome;
    }
    get nome() {
        return this._nome;
    }
    set nome(novoNome) {
```

```
        this._nome = novoNome;
    }
}
const pessoa = new Pessoa('Maria');
console.log(pessoa.nome); // Saída: Maria
pessoa.nome = 'Joana';
console.log(pessoa.nome); // Saída: Joana
```

Herança

Herança permite que uma classe herde propriedades e métodos de outra.

Exemplo:

```
class Animal {
    constructor(tipo) {
        this.tipo = tipo;
    }
}
class Cachorro extends Animal {
    constructor(nome) {
        super('Cachorro');
        this.nome = nome;
    }
}
```

```
const rex = new Cachorro('Rex');

console.log(rex.tipo, rex.nome); // Saída: Cachorro Rex
```

InstanceOf

`instanceof` verifica se um objeto pertence a uma classe.

Exemplo:

```
console.log(rex instanceof Cachorro); // Saída: true

console.log(rex instanceof Animal);  // Saída: true
```