Conceitos Fundamentais de Orientação a Objetos

Programação Orientada a Objeto (POO) é um paradigma de projeto e programação de *software* baseado na abstração digital do mundo real, através da composição e interação entre diversas unidades chamadas de **objetos** e as **classes,** representando objetos reais contendo identidade, **atributos** (propriedades) e **métodos** (funções).

A Programação Orientada a Objetos se baseia em quatro principais componentes da programação: **abstração**, **encapsulamento**, **herança** e **polimorfismo**.

Classes
Classes são modelos ou templates a partir dos quais objetos são criados.
Exemplo: Classe **Carro** com propriedades **cor** e **marca**, e método dirigir.

Objetos
Objetos são **instâncias** ou cópias de classes que possuem atributos (propriedades) e métodos (comportamentos).
Exemplo: Um carro com propriedades cor: “branca”, marca: “Honda” e método dirigir.

Atributos
São as características ou propriedades de um objeto.
Exemplo: A cor de um carro.

Métodos
São as ações ou comportamentos de um objeto.
Exemplo: Dirigir um carro.

Abstração
A abstração oculta detalhes complexos e mostra apenas as características essenciais de um objeto ou sistema, permitindo que os desenvolvedores se concentrem no que um objeto faz, em vez de como ele faz. A abstração é alcançada definindo classes que representam tipos abstratos de comportamento e características. Por exemplo, uma classe "Carro" pode incluir atributos como "marca", "modelo" e "cor", e métodos como "ligar" e "acelerar", sem detalhar os mecanismos internos dessas funções.

Encapsulamento
Esconder os detalhes internos de um objeto e expor apenas o necessário.
Exemplo: Propriedades privadas e métodos públicos.

Herança
Permitir que uma classe herde propriedades e métodos de outra classe.
Exemplo: Classe Carro que herda de uma classe Veiculo.

Polimorfismo
Capacidade de um método assumir diferentes formas.
Exemplo: Método dirigir funciona de forma diferente para uma moto e um carro.

1. **Criar uma classe em JavaScript.**

```
class Carro {
    constructor(_marca, _modelo) {
        this.modelo = _modelo;
        this.marca = _marca;
    }

    // Método público
    dirigir() {
        console.log(`Faça um Test Drive no novo ${this.marca} ${this.modelo}.`);
    }
}

const carro = new Carro("Honda", "Civic");
carro.dirigir();
```

2. **Encapsulamento**

- A partir do ECMAScript 2022, foi introduzido campos privados com a notação #.

```
class Carro {
    #cor; // Propriedade privada

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

    // Método público
    dirigir() {
        console.log(`O carro da marca ${this.marca} e cor ${this.#cor}.`);
    }

    // Getter e Setter para propriedade privada
    get cor() {
        return this.#cor;
    }

    set cor(novaCor) {
        this.#cor = novaCor;
    }
}

const meuCarro = new Carro('Honda');
meuCarro.dirigir(); // O carro da marca Honda e cor undefined.
meuCarro.cor = 'azul';
meuCarro.dirigir(); // O carro da marca Honda e cor azul.
```

3. Herança

```
class Veiculo {
    constructor(_marca, _modelo) {
        this.marca = _marca;
        this.modelo = _modelo;
    }

    mostrarVeiculo(_tipo) {
        console.log(`Tipo de veículo: ${_tipo}.`);
    }
}

class Carro extends Veiculo {
    constructor(marca, modelo) {
        super(marca, modelo);
    }

    // Método público
    dirigir() {
        console.log(`Faça um Test Drive no novo ${this.marca} ${this.modelo}.`);
    }
}

const carro = new Carro("Honda", "Civic");
carro.mostrarVeiculo("carro"); // Usa a classe mãe (Veiculo)
carro.dirigir();    // Usa a classe filha (Carro)
```

4. Polimorfismo

```
class Veiculo {
    mover() {
      console.log('O veículo está se movendo.');
    }
  }

  class Carro extends Veiculo {
    mover() {
      console.log('Dirigindo o carro.');
    }
  }

  class Moto extends Veiculo {
    mover() {
      console.log('Pilotando a moto.');
    }
  }
```

```
const carro = new Carro();
const moto = new Moto();

carro.mover(); // Dirigindo o carro.
moto.mover(); // Pilotando a moto.
```

Exercícios

1. Criação de Classe

- Crie uma classe Pessoa
- Propriedades: nome, idade
- Método: falar (exibe "Meu nome é [nome] e eu tenho [idade] anos.")

2. Encapsulamento

- Modifique a classe Pessoa para ter a propriedade idade como privada.
- Adicione getters e setters para idade.

3. Herança

- Crie uma classe Aluno que herda de Pessoa.
- Adicione uma propriedade curso.
- Adicione o método estudar que exibe "O aluno [nome] está estudando [curso]."

4. Polimorfismo

- Crie uma classe Professor que herda de Pessoa.
- Adicione um método trabalhar que exibe "O professor [nome] está dando aula."
- Crie uma instância de Aluno e Professor e chame os métodos falar, estudar e trabalhar.