■ Roteiro de Ensino: POO com TypeScript Aplicada ao Frontend

Este roteiro foi desenvolvido para o ensino dos conceitos de Programação Orientada a Objetos (POO) aplicados ao desenvolvimento Frontend com TypeScript. O conteúdo foca em como estruturar componentes, controlar estado, reutilizar código e consumir APIs de forma organizada.

■ Aula 1 – Classes e Componentização no Frontend

Objetivo: Entender como usar classes para estruturar elementos da interface.
Tópicos: Classe como unidade de componente, atributos para estado e métodos para interação.

```
class Botao {
  private elemento: HTMLButtonElement;

  constructor(private texto: string, private cor: string) {
    this.elemento = document.createElement("button");
    this.elemento.textContent = this.texto;
    this.elemento.style.backgroundColor = this.cor;
    document.body.appendChild(this.elemento);
  }

  onClick(callback: () => void) {
    this.elemento.addEventListener("click", callback);
  }
}

const botaoEnviar = new Botao("Enviar", "#007bff");
botaoEnviar.onClick(() => alert("Enviado com sucesso!"));
```

Desafio: Crie uma classe *Card* que renderiza um título e uma descrição no DOM, com um botão “Ver mais”.

■ Aula 2 – Encapsulamento e Estado do Componente

Objetivo: Proteger e gerenciar o estado interno dos componentes.
Tópicos: Uso de *private* e *protected*, métodos públicos e atualização de interface.

```
class Contador {
  private valor: number = 0;
  private elemento: HTMLDivElement;

  constructor() {
    this.elemento = document.createElement("div");
    this.render();
    document.body.appendChild(this.elemento);
```

```
  }

  incrementar() {
    this.valor++;
    this.render();
  }

  decrementar() {
    this.valor--;
    this.render();
  }

  private render() {
    this.elemento.innerHTML = `
      <h3>Contador: ${this.valor}</h3>
      <button id="inc">+</button>
      <button id="dec">-</button>
    `;
    this.elemento.querySelector("#inc")?.addEventListener("click", () => this.incrementar());
    this.elemento.querySelector("#dec")?.addEventListener("click", () => this.decrementar());
  }
}

new Contador();
```

Desafio: Crie um componente *ToggleButton* que muda de “Ligado” para “Desligado” quando clicado, mantendo o estado privado.

■ Aula 3 – Herança e Reutilização de Comportamento

Objetivo: Criar componentes que herdam e estendem comportamentos.
Tópicos: Componentes base, herança com *extends* e especialização visual e funcional.

```
class ComponenteBase {
  protected elemento: HTMLElement;

  constructor(tag: string) {
    this.elemento = document.createElement(tag);
  }

  render(container: HTMLElement) {
    container.appendChild(this.elemento);
  }
}

class Alerta extends ComponenteBase {
  constructor(private mensagem: string, private tipo: "sucesso" | "erro") {
    super("div");
    this.elemento.textContent = this.mensagem;
    this.elemento.style.padding = "10px";
    this.elemento.style.color = this.tipo === "sucesso" ? "green" : "red";
  }
```

```
}

const alerta = new Alerta("Operação concluída!", "sucesso");
alerta.render(document.body);
```

Desafio: Crie uma classe *MensagemTemporaria* que herda de *Alerta* e desaparece após 3 segundos.

■ Aula 4 – Abstração, Interfaces e Serviços

Objetivo: Modelar dados e serviços que interagem com o frontend.
Tópicos: Interfaces, serviços de API e separação de responsabilidades (modelo, serviço, componente).

```
interface Usuario {
  id: number;
  nome: string;
  email: string;
}

class UsuarioService {
  async listar(): Promise<Usuario[]> {
    const resposta = await fetch("https://jsonplaceholder.typicode.com/users");
    return resposta.json();
  }
}

class ListaUsuarios {
  constructor(private service: UsuarioService) {}

  async render() {
    const usuarios = await this.service.listar();
    const ul = document.createElement("ul");
    usuarios.forEach(u => {
      const li = document.createElement("li");
      li.textContent = `${u.nome} - ${u.email}`;
      ul.appendChild(li);
    });
    document.body.appendChild(ul);
  }
}

new ListaUsuarios(new UsuarioService()).render();
```

Desafio: Crie uma interface *Produto* e uma classe *ProdutoService* que busque dados falsos e exiba em cards.

■ Mini Projeto Integrador – Dashboard de Loja

Implemente um pequeno **dashboard de loja** utilizando POO em TypeScript. Classe **Produto**: representa o modelo de dados. Classe **ProdutoService**: fornece acesso aos dados (simulação de API). Classe **ProdutoCard**: exibe as informações visualmente.

Classe **ListaProdutos**: renderiza múltiplos cards e permite filtragem. O objetivo é integrar herança, encapsulamento, abstração e interfaces para construir uma aplicação frontend organizada.