THAMIRIS ADRIANO

PÓS TECH

FULL STACK DEVELOPMENT

TYPESCRIPT

AULA 03

SUMÁRIO

O QUE VEM POR AÍ? 3

SAIBA MAIS 5

O QUE VOCÊ VIU NESTA AULA? 11

REFERÊNCIAS 12

O QUE VEM POR AÍ?

Nessa aula, abordaremos o assunto “Interfaces e Generics em TypeScript”, explicando como esses conceitos são cruciais para criar códigos mais organizados e flexíveis. Com interfaces, você aprenderá a estabelecer padrões claros para estruturas de dados e classes, enquanto Generics ensinarão a trabalhar com vários tipos de dados de forma segura e eficaz. Este conhecimento é fundamental para qualquer profissional de desenvolvimento que deseje aprofundar suas habilidades em TypeScript.

HANDS ON

No primeiro vídeo, os(as) professores(as) abordaram interfaces em TypeScript, essenciais para uma programação mais estruturada e segura. Através de exemplos práticos, demonstraram como interfaces garantem que objetos sigam uma estrutura definida. Ressaltaram também a importância das interfaces em projetos de software, enfatizando sua contribuição para a consistência, segurança e escalabilidade do código.

No segundo vídeo, os(as) professores(as) focaram em Generics, destacando como eles ampliam a flexibilidade e reutilização do código. Começaram explicando o básico dos Generics, mostrando como permitem a criação de funções e classes que trabalham com diversos tipos de dados, e abordaram também a segurança de tipo proporcionada pelos Generics, com exemplos práticos de seu uso em arrays e classes. Por fim, enfatizaram a importância dos Generics no desenvolvimento de software robusto e flexível em TypeScript.

SAIBA MAIS

Interface

Imagine que você tem um controle de videogame em suas mãos. Esse controle é equipado com vários botões, como os de pular, correr, atacar e outros comandos específicos para ações no jogo. Cada botão no controle representa uma "promessa" de que uma ação específica será executada no jogo quando você o pressionar. No entanto, o controle em si não explica como essas ações são realizadas no jogo, ele apenas fornece os botões que você pode usar.

Figura 1 — Ilustração de controle para jogos eletrônicos
Fonte: Freepik (2020)

Em programação, uma interface funciona de maneira semelhante a esse controle de videogame. Ela é como uma coleção de botões ou métodos que uma classe pode ter. Quando uma classe implementa uma interface, ela está basicamente dizendo: "Eu vou ter todos esses botões (métodos) e vou definir o que cada um faz".

Mas a interface, por si só, não define como esses botões (métodos) funcionam internamente, ela apenas especifica quais devem estar presentes. Assim, uma interface age como um contrato ou um conjunto de regras. Ela assegura que diferentes classes que implementam a mesma interface terão um conjunto comum de métodos.

No entanto, cada classe tem a liberdade de implementar esses métodos da maneira que achar mais adequada. Por exemplo, se a interface tem um método chamado "atacar", uma classe pode implementar esse ataque como um soco, enquanto outra classe pode interpretá-lo como um disparo de energia.

Essa abordagem traz uma grande vantagem: promove um design de software consistente e organizado. Os desenvolvedores podem contar com um conjunto padronizado de métodos em diferentes classes, facilitando a integração e o uso dessas classes em diversos contextos. Além disso, a interface permite que o código seja mais flexível e fácil de modificar, permitindo que você altere a forma como uma classe implementa um método sem precisar mudar a interface em si ou outras classes que dependem dela.

Da mesma forma que um controle de videogame oferece uma interface padrão para interagir com um jogo, as interfaces de programação oferecem um meio padronizado e organizado para que diferentes partes do seu código se comuniquem e interajam entre si.

Classes abstratas e interfaces

Quando você explora a programação, percebe que as classes abstratas e interfaces são como ferramentas para tornar seu código mais eficiente e organizado.

Classes abstratas: pense em uma classe abstrata como uma receita básica de bolo. Essa receita define ingredientes essenciais, como farinha e ovos, mas não especifica sabores ou decorações. Da mesma forma, numa classe abstrata, você define características comuns (como "ter rodas", para veículos) que servem de base para outras classes. Por exemplo, um bolo de chocolate e um bolo de baunilha seguem a receita básica, mas adicionam seus próprios sabores e toques especiais.

Interfaces: as interfaces são como checklists para tarefas. Imagine, por exemplo, que uma interface é como uma lista de itens que você precisa levar para uma viagem. Essa lista pode incluir coisas como "roupas", "documentos", "dinheiro", etc. A lista em si não fornece as roupas ou o dinheiro, mas diz o que você precisa levar. Aplicando isso à programação, temos:

Interface como lista de itens (propriedades): uma interface em TypeScript é como essa lista de viagem. Por exemplo, se você tem uma interface chamada Carro, ela pode listar itens como marca, modelo e ano. Assim como a lista de viagem, a interface Carro não diz exatamente o que é cada item (como é a marca ou o modelo), apenas que esses itens devem existir em qualquer "carro" que você criar no seu programa.

Usando a interface em variáveis (tipos): assim como você pode usar a lista de viagem para preparar diferentes malas para diferentes viagens, você pode usar a interface Carro para criar diferentes variáveis do tipo Carro no seu programa. Cada carro terá uma marca, um modelo e um ano, mas os detalhes específicos podem variar. Por exemplo, um carro pode ser um "Celta 2004" e outro um "Ônix 2016".

Estendendo interfaces (compondo tipos mais complexos): imagine que sua lista seja para uma viagem de negócios. Você pode ter uma lista básica para qualquer viagem e, para negócios, você adiciona itens extras, como "laptop" e "relatórios de negócios". Da mesma forma, você pode ter uma interface básica Carro e criar uma nova interface CarroDeLuxo que adiciona itens, como sistemaDeSom e interiorDeCouro, estendendo a interface Carro.

Propriedades opcionais: na sua lista de viagem, você pode colocar itens opcionais, como "câmera", por exemplo. Isso significa que você pode ou não levar uma câmera, mas é bom ter a opção. Em uma interface, você também pode ter propriedades opcionais. Na interface Carro, você pode ter uma propriedade opcional como tetoSolar. Isso significa que alguns carros podem ter teto solar, mas não é obrigatório para todos.

Métodos em interfaces: imagine que, além de itens, sua lista de viagem também inclui ações que você precisa fazer, como "verificar passagens" ou "reservar hotel". De forma semelhante, nas interfaces, além de propriedades, você pode ter métodos. Por exemplo, na interface Carro, você pode ter um método ligar(), que cada carro implementará de seu jeito.

Portanto, uma interface em TypeScript é como uma lista de verificação para criar objetos: ela define o que deve ser incluído, mas não como esses itens são feitos ou como as ações são realizadas. Cada objeto (ou variável) que usa essa interface precisa seguir as regras da lista, mas pode fazê-lo à sua maneira.

Uso de interfaces no TypeScript

No TypeScript, as interfaces desempenham um papel crucial na estruturação do código e na garantia de que as classes e objetos sigam um padrão específico. Uma interface no TypeScript pode ser usada para definir a forma de um objeto, especificando os tipos de suas propriedades e os métodos que deve implementar.

Definição básica de uma interface:

```
interface Carro {
    marca: string;
    modelo: string;
    ano: number;
    ligar(): void;
}
```

Neste exemplo, a interface Carro define um contrato para qualquer objeto que pretenda ser um carro. Esse contrato inclui propriedades como marca, modelo e ano, além de um método ligar.

Implementando uma interface:

```
class MeuCarro implements Carro {
    marca: string;
    modelo: string;
    ano: number;

    constructor(marca: string, modelo: string, ano: number) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }

    ligar() {
        console.log(`Ligando ${this.modelo}`);
    }
}
```

Aqui, a classe MeuCarro implementa a interface Carro. Isso significa que MeuCarro deve ter todas as propriedades e métodos definidos na interface Carro.

Generics

Imagine que você tem uma caixa mágica que pode mudar seu tamanho e forma para guardar diferentes tipos de coisas. Um dia, você pode usá-la para guardar maçãs, e ela se ajusta ao tamanho e forma das maçãs. No outro dia, você pode usá-la para guardar livros, e ela se ajusta novamente para acomodar os livros. Essa caixa mágica é como os Generics em programação.

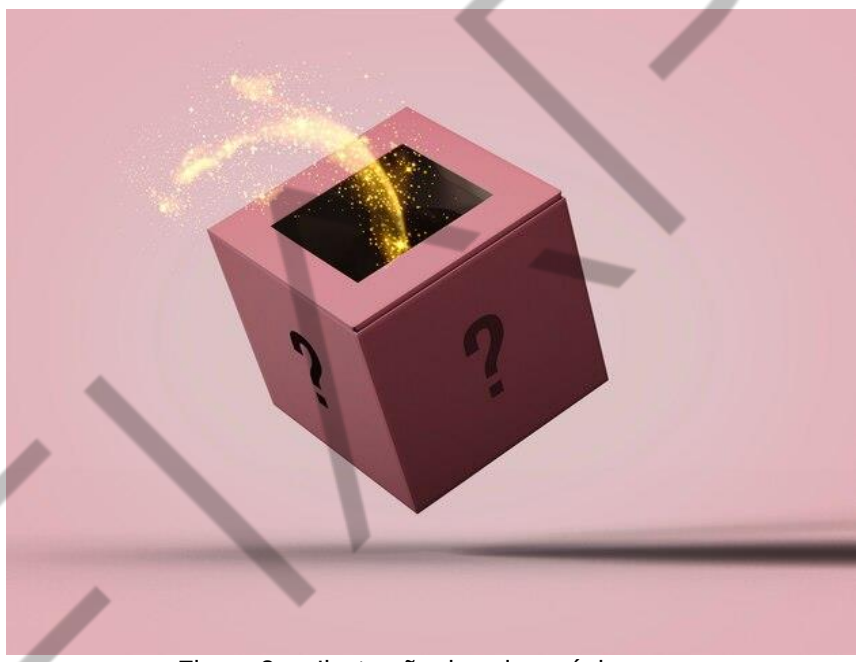

Figura 2 — Ilustração de caixa mágica

Fonte: Freepik (2022)

A seguir, vamos explorar os conceitos essenciais sobre Generics.

Flexibilidade: Generics permitem que você crie componentes (como funções, classes, interfaces) que funcionam com vários tipos de dados, não apenas um. É como ter uma caixa que pode se ajustar para guardar qualquer coisa, em vez de ter uma caixa só para maçãs e outra só para livros.

Uso em TypeScript: em TypeScript, você cria um Generic usando <T>. Esse T é como um espaço reservado para o tipo de dado que você vai usar mais tarde. Por exemplo:

```
function caixaMagica<T>(item: T): T {
    return item;
}
```

Aqui, caixaMagica é uma função que pode trabalhar com qualquer tipo de dado. T é o tipo que você decidirá quando usar a função.

Exemplos Práticos:

- Se você usar caixaMagica<number>(5), o T se torna um número e a função funciona com números.
- Se usar caixaMagica<string>('Olá'), agora o T é uma string e a função lida com strings.

Mais de um tipo: você também pode fazer uma caixa mágica que lida com mais de um tipo de coisa ao mesmo tempo. Por exemplo:

```
function caixaDupla<T, U>(item1: T, item2: U): [T, U] {
    return [item1, item2];
}
```

Aqui, caixaDupla pode guardar dois tipos diferentes de coisas ao mesmo tempo, como um número e uma string.

Em resumo, Generics em TypeScript são como ferramentas flexíveis que você pode ajustar para trabalhar com diferentes tipos de dados. Eles são úteis porque permitem que você escreva código mais geral e reutilizável, assim como uma caixa mágica que pode se ajustar para guardar qualquer coisa.

O QUE VOCÊ VIU NESTA AULA?

Durante essa aula, focamos em conceitos essenciais da programação em TypeScript: interfaces, classes abstratas e Generics. Compreendemos que interfaces são como contratos para as classes, estabelecendo quais métodos devem ser implementados, mas sem ditar a implementação específica. As classes abstratas servem como base para outras classes, fornecendo uma estrutura comum e permitindo variações. Já os Generics oferecem flexibilidade, permitindo a criação de componentes que podem trabalhar com diversos tipos de dados. Esse conhecimento é crucial para desenvolver código mais organizado, reutilizável e fácil de manter, elementos fundamentais para uma programação eficiente e robusta.

REFERÊNCIAS

ADRIANO, T. **Guia prático de TypeScript: Melhore suas aplicações JavaScript.** São Paulo: Casa do Código, 2021.

FELIX, R. **Programação Orientada a Objetos.** [s.l.]: Editora Pearson, 2016.

DevMedia. **Orientação a Objetos - simples assim!**. Disponível em: <https://www.devmedia.com.br/orientacao-a-objetos-simples-assim/3254>. Acesso em: 24 jan. 2024.

PALAVRAS-CHAVE

Palavras-chave: Typescript. Desenvolvimento Básico. Superset.