03

Interface de método

Transcrição

Em TypeScript, uma classe só pode herdar de uma classe apenas. Sendo assim, já gastamos essa possibilidade ao herdarmos de `Imprimivel`. No entanto, não herdamos código nenhum, porque `Imprimivel` é uma classe abstrata que define apenas um método abstrato, diferente da classe `View` que definimos como abstrata, mas que possui outros métodos concretos para serem reutilizados. Em suma, lançamos mão da herança para ganharmos polimorfimo e para podemos obrigar as classes `Negociacao` e `Negociacoes` a implementarem o método `paraTexto()`. Quando queremos apenas obrigar classes a seguirem um método contrato de método, podemos também trabalhar com interfaces. Já trabalharmos com interfaces antes, vamos trabalhar com ela agora de outra forma.

Vamos transformar `Imprimivel` em uma interface que define a obrigatoriedade de implementarmos o método `paraTexto()`:

```
export interface Imprimivel {

    paraTexto(): void;
}
```

Mudamos de `class` para `interface`. Além disso, não é necessário adicionar o modificador `abstract`, pois todos os métodos de uma interface são abstratos por natureza, não possuem uma implementação, um corpo.

Agora, vamos fazer com que `Negociacao` e `Negociacoes` implementes a inteface `Imprimivel`:

```
import { Imprimivel } from './Imprimivel';

export class Negociacao implements Imprimivel {

    constructor(readonly data: Date, readonly quantidade: number, readonly valor: number) {}

    get volume() {

        return this.quantidade * this.valor;
    }

    paraTexto(): void {
        console.log('-- paraTexto --');
        console.log(
            `Data: ${this.data}
            Quantidade: ${this.quantidade},
            Valor: ${this.valor},
            Volume: ${this.volume}`
        )
    }
}
```

Veja que não há mais a necessidade de usarmos `super()`, pois não estamos mais estendendo outra classe.

Alterando `Negociacoes` agora:

```
import { Negociacao } from './Negociacao';
import { Imprimivel } from './Imprimivel';


export class Negociacoes implements Imprimivel {

    private _negociacoes: Negociacao[] = [];

    adiciona(negociacao: Negociacao): void {

        this._negociacoes.push(negociacao);
    }

    paraArray(): Negociacao[] {

        return ([] as Negociacao[]).concat(this._negociacoes);
    }

    paraTexto(): void {
        console.log('-- paraTexto --');
        console.log(JSON.stringify(this._negociacoes));
    }
}
```

Não precisamos alterar a função `imprime` , pois o tipo continua sendo `Imprimivel[]` , mas em vez de `Imprimivel` ser uma classe, agora é uma interface.

Nosso código continua funcionando, excelente. Deixamos de herdar desnecessariamente de uma classe e garantimos um comportamento comum entre objetos através de uma interface. Contudo, há algo curioso.

Não é necessário que uma classe implemente a interface para que seja um valor válido. A implementação de uma interface não deixa o desenvolvedor esquecer de implementar o método, mas podemos fazer algo assim que é totalmente válido:

```
imprime(negociacao, this._negociacoes, { paraTexto: () => console.log('oi')});
```

Veja que o terceiro parâmetro é um objeto JavaScript que possui a função `paraTexto` com a mesma assinatura da interface `Imprimivel` . TypeScript é inteligente e quando encontra uma paridade com o tipo de interface compila o código. Basta trocarmos uma letrinha por exemplo, `paraText` para que o código não compile.