02

Polimorfismo

Transcrição

Uma solução é criarmos a classe `Imprimivel` para que seja herdada por `Negociacao` `Negociacoes`. Como `Negociacao` é um `Imprimivel` e `Negociacoes` também, podemos referenciar instâncias dessas classes através do tipo `Imprimivel`. Como a super classe não sabe de antemão qual a implementação do método `paraTexto()`, vamos torná-lo um método abstrato, aquele que obriga a classe filha implementá-lo.

```
// app/ts/models/Imprimivel.ts

export abstract class Imprimivel {

    abstract paraTexto(): void;
}
```

```
export * from './Negociacao';
export * from './Negociacoes';
export * from './NegociacaoParcial';
export * from './Imprimivel';
```

Agora, só precisamos estender a classe `Imprimivel` em `Negociacao` e `Negociacoes`. Como ambas já possuem o método `paraTexto()` com a mesma assinatura (forma) do método abstrato, não teremos erro de compilação pois já atendemos a obrigação de implementar o método:

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

export class Negociacao extends Imprimivel {

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

        super();
    }

    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 bastou apenas estender `Imprimivel`, como o constructor da classe filha é diferente da classe pai, fomos obrigados a chamar `super()` no constructor da classe filha para que o constructor da classe pai seja chamado.

Agora, faremos a mesma coisa com `Negociacoes`:

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

export class Negociacoes extends 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));
    }
}
```

Neste caso, não precisamos chamar `super()`, porque a classe filha não define um constructor e, nesse caso, por padrão, o TypeScript adotará o constructor da classe pai.

Por fim, basta alterarmos agora nossa função `impressão` para que trabalhe com um array do tipo `Imprimivel[]`:

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

export function imprime(...objetos: Imprimivel[]) {
    objetos.forEach(objeto => objeto.paraTexto());
}
```

Veja que o autocomplete funciona, mostrando apenas os métodos disponíveis na classe pai, omitindo os métodos classe filha. Não há problema nenhum nisso, porque queremos apenas lidar com objetos que são imprimíveis, e basta eles terem o método `paraTexto()`.

Agora, em `NegociacaoController`, nosso código não compilará, porque `Date` não é um `Imprimivel`:

```
Argument of type 'Date' is not assignable to parameter of type 'Imprimivel'.
```

Ao removermos o parâmetro inválido (que não fazia mesmo sentido passarmos) nosso código volta a compilar. Lançamos mão do polimorfismo que consiste em tratar instâncias de objetos de diferentes formas. Sendo assim, apesar de na memória termos objetos do tipo `Negociacao` e `Negociacoes`, eles são referênciados através do tipo `Imprimivel`. Contudo, nossa solução ainda não esta 100% e precisamos meditar sobre ela antes de continuarmos.