02

Herança

Transcrição

Se compararmos os arquivos NegociacoesView e MensagemView lado a lado, veremos que ambos são praticamente parecidos. Por exemplo, ambos possuem a propriedade `_elemento` e um `constructor()` que recebem uma string.

Vamos isolar esse código repetido dentro da classe `View`, que declararemos no arquivo `app/ts/views/View.ts`:

```
class View {

    private _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }
}
```

Agora, vamos remover a propriedade `_elemento` e o `constructor()` de `MensagemView` e `NegociacaoView`. Com certeza ficaremos com erro de compilação.

Agora, utilizando um conceito do paradigma orientado a objetos, vamos fazer com que `NegociacaoView` e `MensagemView` herdem o código que centralizamos em `View`. Fazemos isso através da instrução `extends`.

```
class NegociacoesView extends View{

    update(model: Negociacoes) {

        // erro de compilação
        this._elemento.innerHTML = this.template(model);
    }
    // código posterior omitido


class MensagemView extends View {

    update(model: string) {

        // erro de compilação
        this._elemento.innerHTML = this.template(model);
    }

    // código posterior omitido
```

Quando usamos a instrução `extends`, podemos fazer com que uma classe herde de outra suas propriedades e métodos, inclusive `constructor()`, no entanto nosso código não compila. Recebemos a mesma mensagem de erro para `NegociacoesView` e `MensagemView`:

```
Property '_elemento' is private and only accessible within class 'View'.
```

O problema é que propriedades declaradas com o modificador `private` não podem ser acessadas nem mesmo pelas classes filhas. Podemos resolver esse erro de compilação relaxando um pouco o nível de acesso à propriedade, utilizando o modificador `protected` em vez de `private`:

```
class View {

    // mudou para protected!
    protected _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }
}
```

Após isto adicionaremos o arquivo View.js no script do HTML.

```
<script src="js/models/Negociacao.js"></script>
    <script src="js/controllers/NegociacaoController.js"></script>
    <script src="js/models/Negociacoes.js"></script>
    <script src="js/views/View.js"></script>
    <script src="js/views/NegociacoesView.js"></script>
    <script src="js/views/MensagemView.js"></script>
    <script src="js/app.js"></script>
```

Agora, automaticamente, o VSCode deixará de exibir as mensagens de erro. Inclusive, podemos recarregar nossa página e testar nossa aplicação, que continuará funcionando.

Mas será que podemos reutilizar mais código?