

TS TypeScript – Herança

TypeScript também fornece suporte a herança (um conceito bem conhecido de OO, basicamente é você fazer objetos filhos herdarem coisas de alguma classe que é seu pai).

Então, imagina a seguinte situação, eu tenho dois módulos **TypeScript** de **view** (elemento visual/visualização), essas **views** tem um **elemento de HTML como atributo** que é **recebido através de um seletor pelo construtor**, essas **views** então **pegam esse elemento e colocam um template HTML dentro deles** (com alguma **interpolação de valor**):

```
export class MensagemView {
    private elemento : HTMLElement

    constructor(seletor : string) {
        this.elemento = document.querySelector(seletor);
    }

    template(model : string) : string {
        return `
        <p class="alert alert-info"> ${model}</p>
        `
    }

    atualizar(model : string) : void {
        const template = this.template(model);
        this.elemento.innerHTML = template;
    }
}

export class NegociacoesView {
    private elemento : HTMLElement

    constructor(seletor : string) {
        this.elemento = document.querySelector(seletor);
    }

    template(model : Negociacoes) : string {
        return `
        <table class="table table-hover table-bordered">
            <thead>
                <tr>
                    <th>DATA</th>
                    <th>QUANTIDADE</th>
                    <th>VALOR</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>10/06/2022</td>
                    <td>2</td>
                    <td>133.67</td>
                </tr>
            </tbody>
        </table>
        `
    }

    atualizar(model : Negociacoes) : void {
        const template = this.template(model);
        this.elemento.innerHTML = template;
    }
}
```

Basicamente a ideia disso tudo é criar um componente **visual HTML** para **renderizar na tela um determinado valor**, que estão sendo chamadas em outro **módulo TS**:

```
export class NegociacaoController {
    private inputData : HTMLDataElement;
    private inputQuantidade : HTMLInputElement;
    private inputValor : HTMLInputElement;

    private negociacoes = new Negociacoes();

    private negociacoesView = new NegociacoesView('#negociacoesView');
    private mensagemView = new MensagemView('#mensagemView');

    <div id="mensagemView"></div>
    <div id="negociacoesView">

```

Negociações

Negociação adicionada com sucesso!

Data
dd/mm/aaaa

Quantidade

Valor

Incluir

DATA	QUANTIDADE	VALOR
10/06/2022	2	133.67

Então onde entra a **herança** nisso? Bom, eu posso **reaproveitar o código repetido utilizando uma classe pai**, já que minhas **views “especializadas”** possuem um **elemento**, eu vou delegar a uma classe de **View genérica** que **ela tenha esse código em comum** e as **filhas** apenas **o aproveitem estendendo-a**:

```
export class NegociacoesView {
    private elemento : HTMLElement

    constructor(seletor : string) {
        this.elemento = document.querySelector(seletor);
    }
}

export class MensagemView {
    private elemento : HTMLElement

    constructor(seletor : string) {
        this.elemento = document.querySelector(seletor);
    }
}

export class View {
    protected elemento : HTMLElement

    constructor(seletor : string) {
        this.elemento = document.querySelector(seletor);
    }
}

export class MensagemView extends View {
    template(model : string) : string {
        return `
        <p class="alert alert-info"> ${model}</p>
        `
    }
}

export class NegociacoesView extends View {
    template(model : Negociacoes) : string {
        return `
        <table class="table table-hover table-bordered">
            <thead>
                <tr>
                    <th>DATA</th>
                    <th>QUANTIDADE</th>
                    <th>VALOR</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>10/06/2022</td>
                    <td>2</td>
                    <td>133.67</td>
                </tr>
            </tbody>
        </table>
        `
    }
}
```

O elemento precisa ser **protected** pois nesse caso as classes filhas precisam poder alterar seu valor.

TS TypeScript – Herança com Genéricos

Outra repetição de código que existe nas **views especializadas** é a **atualização do template**, mas tem algo aqui, por mais que o código seja idêntico em questão de corpo eles **são diferentes no parâmetro que recebem**, tanto o **template** quanto a **atualização do template**:

```
export class NegociacoesView extends View {
  template(model : Negociacoes) : string { ... }

  atualizar(model : Negociacoes) : void {
    const template = this.template(model);
    this.elemento.innerHTML = template;
  }
}

export class MensagemView extends View {
  template(model : string) : string {
    return `
      <p class="alert alert-info"> ${model}</p>
    `;
  }

  atualizar(model : string) : void {
    const template = this.template(model);
    this.elemento.innerHTML = template;
  }
}
```

Basta utilizar **Generics**, no **View pai** eu vou dizer que ele pode se aplicar a qualquer tipo com **"<T>"**, nas **filhas** eu **especifico esse tipo**:

```
export class View <T> {
  protected elemento : HTMLElement

  constructor(seletor : string){
    this.elemento = document.querySelector(seletor);
  }

  template(model : T) : string { ... }

  atualizar(model : T) : void { ... }
}

export class MensagemView extends View <String> {
  template(model : string) : string {
    return `
      <p class="alert alert-info"> ${model}</p>
    `;
  }
}

export class NegociacoesView extends View <Negociacoes> {
  template(model : Negociacoes) : string { ... }
}
```

TS TypeScript – Classe Abstrata

Uma classe abstrata é uma classe que não pode ser instanciada (*basicamente o papel dela é ser um "molde" pros filhos*), a classe **View** por exemplo, ela é um molde pras **views especializadas** contendo métodos e atributos reaproveitáveis, então faz sentido que ela seja **abstrata**.

Inclusive, o **template que é o método mais especializado** (é o código html que faz a renderização dependendo do objeto que o **view** representa) pode ser **abstrato**, já que o pai não tem a capacidade de implementá-lo, ele é só uma "ideia", então cabe **aos filhos implementarem essa ideia**:

```
export abstract class View <T> {  
    protected elemento : HTMLElement  
  
    constructor(seletor : string){...  
    }  
  
    abstract template(model : T) : string  
  
    atualizar(model : T) : void {...  
    }  
}
```

```
export class MensagemView extends View <String> {  
    template(model : string) : string {  
        return `  
            <p class="alert alert-info"> ${model}</p>  
        `;  
    }  
}
```