Programação Orientada a Objetos (POO) - Aula 11

Professor: Jerffeson Gomes Dutra

Super

Em TypeScript (assim como em outras linguagens orientadas a objetos), a palavra-chave `super` é usada dentro de uma subclasse para chamar métodos e acessar propriedades da classe pai, ou seja, da superclasse. Isso é especialmente útil quando você quer estender o comportamento de uma classe pai em uma classe filha.

A utilização mais comum do `super` é dentro do construtor da subclasse, para chamar o construtor da classe pai. Isso é necessário porque, quando você cria uma instância da subclasse, geralmente deseja inicializar tanto os atributos da classe pai quanto os atributos da classe filha.

Por exemplo, considere as classes **`Veiculo`** e **`Carro`**:

```
class Veiculo {
    constructor(public marca: string, public modelo: string) {}
}

class Carro extends Veiculo {
    constructor(public numeroPortas: number, marca: string, modelo: string) {
        super(marca, modelo); // Chamando o construtor da classe pai (Veiculo)
    }
}
```

Tratamento de Exceções

Em TypeScript, assim como em outras linguagens de programação, o tratamento de exceções é fundamental para lidar com situações imprevistas que podem ocorrer durante a execução do programa.

As exceções são erros ou condições inesperadas que ocorrem durante a execução do código e que podem interromper o fluxo normal do programa.

Tratamento de Exceções

O tratamento de exceções em TypeScript é feito usando os blocos **try, catch e finally**.

- **try:** é usado para envolver o código que pode gerar uma exceção.
- **catch:** é usado para capturar e lidar com a exceção, caso ocorra.
- **finally:** é opcional e é usado para executar código que deve ser executado independentemente de ocorrer ou não uma exceção.

Exemplo

```
try {

    const resultado = 10 / 0; // Tentativa de divisão por zero

    console.log('Resultado:', resultado); // Esta linha não será alcançada devido à exceção

} catch (erro) {

    // Bloco de código para lidar com a exceção
    console.error('Ocorreu um erro:', erro);

} finally {
    // Bloco de código que é executado independentemente de ocorrer uma exceção ou não
    console.log('Finalizando...');
}
```

Lançando exceção

É possível você lançar exceções manualmente.

O TS tem uma exceção genérica chamada **`ERROR`** `que pode ser usada.`

```
throw new Error('Esta é uma mensagem de erro padrão.');
```

Exemplo de lançamento de Exceção

```
function exemplo() {
    throw new Error('Esta é uma mensagem de erro padrão.');
}

try {
    exemplo();
} catch (erro) {
    console.error('Ocorreu uma exceção:', erro.message);
}
```

Criando uma exceção

Em TypeScript (TS) você pode criar uma classe que estende a classe **`Error`** nativa do JavaScript e adicionar os comportamentos específicos que deseja para sua exceção personalizada.

```
class MinhaExcecaoPersonalizada extends Error {

    name: string = 'MinhaExcecaoPersonalizada';
    stack: string | undefined;

    constructor(mensagem: string) {
        super(mensagem);
        this.stack = (new Error()).stack; // Captura a pilha de chamadas
    }
}
```

Exemplo de uso de Exceção personalizada

```
function exemplo() {
    throw new MinhaExcecaoPersonalizada('Esta é uma mensagem de erro
personalizada.');
}

try {
    exemplo();
} catch (erro) {
    console.error('Ocorreu uma exceção:', erro.message);
}
```

Exercício 01

Suponha que você esteja desenvolvendo um sistema de gerenciamento de inventário para um jogo de RPG.

Crie a classe Inventario e a classe Item.

Em inventário, crie uma função chamada **`adicionarItemAoInventario(item: Item, qtd:number)`** que aceite um item e a quantidade como entrada.

Se o item já estiver no inventário, a quantidade deve ser somada à quantidade existente.

Se o item não estiver no inventário, ele deve ser adicionado com a quantidade fornecida.

No entanto, se a quantidade fornecida for negativa, lance uma exceção indicando um erro.

Se a adição do item no inventário for bem sucedida, retornar o inventário, se houver erro, exibir a mensagem: ‘Não foi possível adicionar o item ao inventário’.

Use um bloco **`try-catch-finally`** para lidar com a exceção e garantir que o código finalize corretamente.

Várias exceções

```
try {
    // Código que pode gerar uma exceção
    const resultado = 10 / 0; // Isso lançará uma exceção de divisão por zero
} catch (erro) {
    if (erro instanceof TypeError) {
        console.error('Ocorreu um TypeError:', erro.message);
    } else if (erro instanceof RangeError) {
        console.error('Ocorreu um RangeError:', erro.message);
    } else {
        console.error('Ocorreu um erro desconhecido:', erro);
    }
}
```