Programação Orientada a Objetos (POO) - Aula 14

Professor: Jerffeson Gomes Dutra

Resumo - Objetos

Objetos são instâncias de classes. Enquanto as classes são um molde, os objetos são o produto que sai com esses moldes.
Cada objeto tem suas próprias características.

Resumo - Criando Objetos

A palavra **NEW** é usada para instanciar/criar objetos de uma classes.

```
class Animal {

    constructor() {

    }
}

class Gato {}

class Cachorro {}
```

```
let animal = new Animal("Mingual", 3);

let animal = new Animal();

let cat = new Gato();

let dog = new Cachorro();
```

Resumo - Criando Objetos

A palavra **NEW** é usada para instanciar/criar objetos de uma classes.

```
class Animal {
    nome: string;

    constructor(n: string) {
        this.nome = n;
    }

}
```

```
let animal = new Animal("Mingual");
```

Resumo - Criando Objetos

A palavra **NEW** é usada para instanciar/criar objetos de uma classes.

```
class Animal {
    nome: string;

    constructor(n: string, idade: string) {
        this.nome = n;
    }

}
```

```
let animal = new Animal("Mingual", 3);
```

Métodos

```
class Animal {

    emitrSom() {
        console.log("Fez barulho");
    }

}
```

```
let animal = new Animal();
animal.emitirSom();
```

Métodos - Com Parâmetros

```
class Animal {

    comer(comida: string) {
        console.log("Animal comeu: " + comida);
    }

}
```

```
let gato = new Animal();

let cachorro = new Animal();


gato.comer("leite");
chachorro.comer("carne");
```

Herança

É um conceito usado para organizar e estruturar dados de forma hierárquica, onde as características de uma classe são compartilhadas por uma ou mais subclasses.

Isso é especialmente útil quando há uma relação de "é um" entre diferentes tipos de entidades.

Por exemplo:

- Gato é um Animal;
- Cachorro é um Animal;

- Mago é um Personagem;
- Cavaleiro é um Personagem;

Exemplo em Código

```
class Animal {

    nome: string

}


class Gato extends Animal {

}


class Cachorro extends Animal {

}
```

```
let gato = new Animal();

let cachorro = new Animal();


gato.nome = "Mingual";
chachorro.nome = "Zeus";


console.log(gato.nome)
console.log(cachorro.nome)
```

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...');
}
```

Exercício 01

- Criem 5 classes e 3 objetos para cada classe

Exercício 02

- Para cada classe crie 2 métodos, um com parâmetro e outro sem.
- Chame esses métodos em cada objeto.