



# Classes

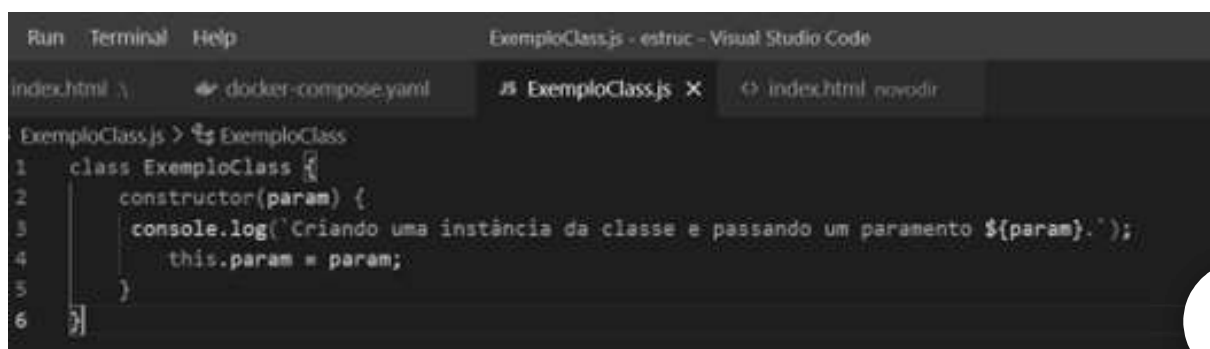
## Resumo

Nesta aula, iremos aprender como utilizar as classes para organizar melhor o nosso desenvolvimento. O uso de classe tem se tornado mais comum no desenvolvimento com JavaScript devido ao crescimento do uso do Node.js. O que você aprenderá aqui será possível utilizar em aplicações, inclusive no Angular. A base que está sendo ensinada permite ser utilizada em outras tecnologias. Portanto, vamos compreender agora o uso de classes.

## Classes

Quando definimos uma classe, um ponto importante é utilizar o construtor. Com ele, é possível manipular os parâmetros que foram passados ao instanciar uma classe.

Ele é definido em um bloco de classe, como se você estivesse definindo um método chamado construtor, embora ele seja realmente manipulado como um caso especial.



```
Run Terminal Help
ExemploClass.js - estruc - Visual Studio Code
index.html \  docker-compose.yaml  JS ExemploClass.js X  index.html novodir
ExemploClass.js > ExemploClass
1 class ExemploClass {
2   constructor(param) {
3     console.log('Criando uma instância da classe e passando um parametro ${param}.');
4     this.param = param;
5   }
6 }
```

Fonte: Autor



Como podemos usar essa classe?

```
const minha = new ExemploClass('teste da classe');  
// imprime: "Criando uma instância da classe e passando um parâmetro teste da classe."
```

Fonte: Autor

Outra forma de usar classe é através de herança. A herança funciona exatamente como em outras linguagens orientadas a objetos: os métodos definidos na superclasse são acessíveis na subclasse de extensão. Se a subclasse declara seu próprio construtor, então ele deve invocar o construtor pai via `super()` antes que ele possa acessar `this`.

```
class Animal{
  constructor(tipo) {
    this.tipo = tipo;
  }

  come(alimento) {
    console.log("Esse animal se alimenta de: " + alimento);
  }
}

class Cachorro extends Animal{
  constructor(tipo) {
    super(tipo);
  }

  latir() {
    console.log("Cachorro late!");
  }
}


class Gato extends Animal{
  constructor(tipo) {
    super(tipo);
  }


  banho() {
    console.log("Gato não toma banho no chuveiro");
  }
}

// OBJETO PITBULL(Cachorro)
let pitbull= new Cachorro();
pitbull.come("ração");
pitbull.latir();
```

Fonte: Autor

Getters e setters permitem que você defina um comportamento personalizado para ler e escrever uma determinada propriedade em sua classe. Para o usuário, eles aparecem como uma propriedade normal. No entanto, internamente, uma função

personalizada é fornecida e usada para determinar o valor quando a propriedade é acessada (o getter) e para execução ; as alterações necessárias quando é atribuída (o setter). Em uma definição de classe, um getter é escrito como um método sem argumento prefixado pela palavra-chave `get`. Um setter é semelhante, exceto que ele aceita um argumento (o novo valor que está sendo atribuído). Aqui está uma classe de exemplo que fornece um getter e setter para sua propriedade `.nome`.



```
1  class Professor {
2
3      constructor(nome, disciplina){
4          this._nome = nome;
5          this._disciplina = disciplina;
6      }
7
8      set nome(value) {
9          this._nome = value;
10     }
11     get nome() {
12         return this._nome;
13     }
14
15     set disciplina(value) {
16         this._disciplina = value;
17     }
18     get disciplina() {
19         return this._disciplina;
20     }
21 }
22
23 const professor1 = new Professor("Marcelo", "JavaScript")
24 console.log("O nome do professor é: " + professor1.nome)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE


TERMINAL

C:\Program Files\nodejs\node.exe .\testeclass.js

O nome do professor é: Marcelo

Fonte: Autoral

## Composição

Composição é um padrão de design em que um objeto é composto por outros objetos menores, cada um responsável  por uma parte específica da funcionalidade geral. Em JavaScript, a composição é uma técnica que permite criar objetos complexos a partir de outros objetos mais simples, combinando suas funcionalidades.

A composição em JavaScript geralmente é realizada por meio de objetos aninhados ou funções que retornam objetos. Um objeto pode ser composto por outros objetos que fornecem comportamentos específicos. Por exemplo, suponha que você tenha uma classe “Carro” que precisa ter recursos de “Motor”, “Transmissão” e “Rodas”. Em vez de criar uma classe grande que implemente tudo isso, você pode criar objetos separados para cada recurso e combiná-los usando a composição.

Veja um exemplo básico de como usar a composição em JavaScript:



```
// Classe Motor
class Motor {
  constructor(cilindrada) {
    this.cilindrada = cilindrada;
  }

  ligar() {
    console.log("Motor ligado");
  }
}

// Classe Carro
class Carro {
  constructor(motor) {
    this.motor = motor;
  }

  acelerar() {
    console.log(`Acelerando com um motor de ${this.motor.cilindrada} cilindradas`);
  }
}

// Criar um objeto Motor
const motor = new Motor(2000);


// Criar um objeto Carro com o objeto Motor como argumento
const carro = new Carro(motor);

// Usar o método acelerar do objeto Carro
carro.acelerar();
```

Fonte: Autoral

Neste exemplo, a classe “Carro” recebe um objeto “Motor” como argumento do construtor e o armazena em uma propriedade. O método “acelerar” do objeto “Carro” usa o objeto “Motor” para obter informações sobre a cilindrada e imprimir uma mensagem. Isso permite que a classe “Carro” tenha acesso a funcionalidade específica do objeto “Motor” sem precisar implementá-la diretamente.

## Conteúdo Bônus

Para utilizar da melhor forma as classes em JavaScript, devemos separar cada classe em um arquivo e transformar em um  Julo que será importado onde desejamos usar. Vejamos o exemplo abaixo:

```
1  module.exports = class Coordenador {  
2      //propriedades e funções da classe  
3      constructor(nome, tempoDeCasa, email) {  
4          this.nome = nome;  
5          this.tempoDeCasa = tempoDeCasa;  
6          this.email = email;  
7          this.dataCadastro = new Date();  
8      }  
9  
10     isSenior(){  
11         return this.tempoDeCasa >= 12;  
12     }  
13  
14     getPrimeiroNome(){  
15         return this.nome.split(" ")[0];  
16     }  
17 }
```

Fonte: Autoral

Para utilizar essa classe em outra classe, temos que fazer o require. Dessa forma, conseguimos usar de uma forma mais organizada o javascript orientado a objetos.



```
//programa.js
const Coordenador = require("../Coordenador");
const coordenador1 = new Coordenador("Marcelo", 16, "marcelo@gmail.com");
const coordenador2 = new Coordenador("Lucas", 5);
console.log(coordenador1.nome + " é Senior? " + coordenador1.isSenior());
console.log(coordenador2.nome + " é Senior? " + coordenador2.isSenior());
```

Fonte: Autoral

Agora podemos chamar o programa e teremos o seguinte resultado:

```
C:\Program Files\nodejs\node.exe .\programa.js
Marcelo é Senior? true
Lucas é Senior? false
```

Fonte: Autoral

## Referência Bibliográfica

FLANAGAN, David. **JavaScript: O Guia Definitivo**. 6ª Ed. Porto Alegre: Bookman, 2013.

FREEMAN, Eric. **Use a cabeça!: programação JavaScript**. 1ª Ed. São Paulo: Alta Books, 2016.

**Ir para exercício**