BACHARELADO EM SISTEMAS DE INFORMAÇÃO
CEFET-RJ NOVA FRIBURGO

Programação de Clientes Web

PROF. THIAGO DELGADO PINTO
thiago.pinto@cefet-rj.br

classes

versão: 2025.09.29

classes

notação de classe

introduzida no ES6 (ES2015)

substitui a sintaxe de funções como classes

- internamente, a representação anterior continua

possui limitações

- não suporta encapsulamento, interfaces, métodos e classes abstratas ou finais, etc.

sintaxe básica


```
class NomeDaClasse {
  atributo1;
  atributo2;
  atributoN;

  constructor( ... ) { ... }
  metodo1( ... ) { ... }
  metodo2( ... ) { ... }
  metodoN( ... ) { ... }
}
```

sintaxe básica


```
class NomeDaClasse {
  atributo1;
  atributo2;
  atributoN;

  constructor( ... ) { ... }
  metodo1( ... ) { ... }
  metodo2( ... ) { ... }
  metodoN( ... ) { ... }
}
```

⚠ NÃO há *let/const/var*

⚠ NÃO há *function*

exemplo 1

```
class Pessoa {
  constructor( nome ) {
    this.nome = nome; // Atributo adicionado dinamicamente
  }
}
const p = new Pessoa( 'Ana' );
console.log( p.nome );



// ☝ Note que atributos e métodos são todos públicos
```

exemplo 2

```
class Pessoa {
  nome = ''; // 👉 Atributo declarado explicitamente (melhor)
  constructor( nome ) {
    this.nome = nome;
  }
}
const p = new Pessoa( 'Ana' );
console.log( p.nome );
```

exemplo 3

```
class Pessoa {
  /** @type {string} */
  nome = ''; // ☝ Documentação de tipo (ainda melhor)

  /** @param {string} nome */
  constructor( nome ) {
    this.nome = nome;
  }
}
const p = new Pessoa( 'Ana' );
console.log( p.nome );
```

👍 **É indicado documentar atributos e métodos.**

👉 Por questões de espaço nos slides, essa documentação será omitida.

exemplo 4

```
class Pessoa {
  nome = '';
  sobrenome = '';
  constructor( nome, sobrenome ) {
    this.nome = nome;
    this.sobrenome = sobrenome;
  }
  nomeCompleto() {
    return this.nome + ' ' + this.sobrenome;
  }
}

const p = new Pessoa( 'Clarice', 'Linspector' );
console.log( p.nomeCompleto() );
```

herança

herança – exemplo 1

```
class Documento {
  valido() { return false; }
}

class CPF extends Documento {
  constructor( numero ) {
    super(); // 👈 Invoca construtor pai. Obrigatório. Deve vir primeiro no construtor!
    this.numero = ( numero || '' ).replaceAll( /[^\d]/g, '' ); // Remove não-numéricos
  }

  valido() { // 👈 Sobrescreve comportamento do método pai
    return this.numero.length === 11 && this.digitosVerificadoresEstaoCorretos();
  }

  digitosVerificadoresEstaoCorretos() { /* ... returna true/false ... */ }
}
```

herança – exemplo 2

```
class Cliente {
  constructor( nome, email ) {
    this.nome = nome;
    this.email = email;
  }
}

class ClientePJ extends Cliente {
  constructor( nome, email, cnpj ) {
    super( nome, email ); // 👈 Invoca para inicializar nome e email
    this.cnpj = cnpj;
  }
}

const clientes = [ new Cliente( 'Ana', 'ana@site.com' ),
  new ClientePJ( 'Acme', 'contato@acme.com', '38.987.303/0001-12' ) ];
```

herança – exemplo 3


```
const MIN_NOME = 2; const MAX_NOME = 100;

class Cliente {
  constructor( nome, email ) { this.nome = nome; this.email = email; }

  validar() {
    if ( typeof this.nome != 'string' ||
      this.nome.length < MIN_NOME || this.nome.length > MAX_NOME
    ) {
      throw new Error( `Nome deve ter de ${MIN_NOME} a ${MAX_NOME}
caracteres.` );
    }
    if ( ! ( this.email instanceof Email ) || ! this.email.valido() ) {
      throw new Error( 'E-mail deve ser válido.' );
    }
  }
}
```

herança – exemplo 3 (continuação)


```
class ClientePJ extends Cliente {
  constructor( nome, email, cnpj ) {
    super( nome, email ); // 👈 Invoca a atribuição da classe pai
    this.cnpj = cnpj;
   }

  validar() {
    super.validar(); // 👈 Invoca o comportamento da classe pai
    if ( typeof this.cnpj != 'string' || this.cnpj.length != 14 ) {
      throw new Error( 'CNPJ deve ter 14 dígitos numéricos.' );
    }
    if ( ! ( this.cnpj instanceof CNPJ ) || ! this.cnpj.valido() ) {
      throw new Error( 'CNPJ inválido.' );
    }
  }
}
```

propriedades

propriedade

uma propriedade é uma simulação de um atributo.

declaração de uma propriedade

uma propriedade é declarada
através de um ou dois métodos especiais,
que devem conter o mesmo nome.

exemplo 1 – propriedade "nome"

```
class Pessoa {
  _nome = ''; // Atributo comum

  // Obter
  get nome() { return this._nome; }

  // Definir
  set nome( valor ) { this._nome = valor; }
}

const p = new Pessoa();
p.nome = 'Ana'; // Invoca o método com prefixo set
console.log( p.nome ); // Invoca o método com prefixo get
```

uso de uma propriedade

deve-se usar uma propriedade

como um atributo,

nunca como um método.

exemplo 1 – propriedade "nome"

```
class Pessoa {
  _nome = ''; // Atributo comum
  get nome() { return this._nome; }
  set nome( valor ) { this._nome = valor; }
}

const p = new Pessoa();
p.nome = 'Ana'; // 👍
console.log( p.nome ); // 👍

p.nome( 'Bia' ); // ⚠️ Erro (não permite como método)
console.log( p.nome() ); // ⚠️ Erro (idem)
```

exemplo 2 – propriedade apenas de leitura

```
class Animal {
  constructor( tipo, apelido ) {
    this._tipo = tipo;
    this._apelido = apelido;
  }
  get tipo() { return this._tipo; }
  get apelido() { return this._apelido; }
}
const a = new Animal( 'Cachorro', 'Rex' );
console.log( a.tipo, a.apelido ); // Cachorro Rex
a.apelido = 'Toby'; // ⚠️ Erro (não permite definir)
```

exemplo 3 – propriedade apenas de leitura

```
class Telefone {
  constructor( ddd, numero ) {
    this.ddd = ddd; this.numero = numero;
  }
  get completo() { return this.ddd + this.numero; }
  get formatado() {
    return `(${this.ddd}) ${this.numero}`;
  }
}
const tel = new Telefone( '22', '988887777' );
console.log( tel.completo ); // 22988887777
console.log( tel.formatado ); // (22) 988887777
tel.completo = '22977776666'; // ⚠️ Erro (não permite definir)
tel.ddd = '21'; // 👍
```

exemplo 4 – propriedade apenas de escrita

```
class Telefone {
  ddd = '', numero = '';
  constructor( numeroCompleto ) {
    this.completo = numeroCompleto; // 👈 Invoca a propriedade
  }
  set completo( valor ) {
    this.ddd = ( valor || '' ).substring( 0, 2 );
    this.numero = ( valor || '' ).substring( 2 );
  }
  get formatado() {
    return `(${this.ddd}) ${this.numero}`;
  }
}
const tel = new Telefone( '22988887777' );
console.log( tel.formatado ); // (22) 988887777
tel.completo = '22977776666'; // 👍
```

encapsulamento privado

sintaxe de declaração privada

acrescentada oficialmente no **EcmaScript 2022**

pode ser usado nos principais navegadores desde 2021

realizada com o caractere **#** (*hashtag*)

👉 o caractere **#** passa a fazer parte do nome

do atributo/método

exemplo 1 – atributo privado

```
class Pessoa {
  #nome = ''; // 👈 Atributo privado
  constructor( nome ) {
    this.#nome = nome; // 👈 Hashtag faz parte do atributo
  }
  getNome() { return this.#nome; }
  setNome( valor ) { this.#nome = valor; }
}

const p = new Pessoa( 'Bia' );
console.log( p.getNome() ); // 👍
p.#nome = 'Beatriz'; // ⚠️ Erro (acesso não permitido)
p.setNome( 'Beatriz' ); // 👍
```

exemplo 2 – atributo privado e propriedade

```
class Pessoa {
  #nome = ''; // 👈 Atributo privado
  constructor( nome ) {
    this.#nome = nome; // 👈 Hashtag faz parte do atributo
  }
  get nome() { return this.#nome; }
  set nome( valor ) { this.#nome = valor; }
}

const p = new Pessoa( 'Bia' );
console.log( p.nome ); // 👍
p.#nome = 'Beatriz'; // ⚠️ Erro (acesso não permitido)
p.nome = 'Beatriz'; // 👍
```

exemplo 3 – método privado

```
class Telefone {
  ddd = '', numero = '';
  constructor( numeroCompleto ) {
    this.#definirPorCompleto( numeroCompleto ); // 👈
  }
  #definirPorCompleto( valor ) { // 👈 método privado
    this.ddd = ( valor || '' ).substring( 0, 2 );
    this.numero = ( valor || '' ).substring( 2 );
  }
  get formatado() {
    return `(${this.ddd}) ${this.numero}`;
  }
}
const tel = new Telefone( '22988887777' );
console.log( tel.formatado ); // (22) 988887777
tel.#definirPorCompleto( '22977776666' ); // ⚠️ Erro (privado)
```

exemplo 4 – propriedade privada

```
class Telefone {
  ddd = '', numero = '';
  constructor( numeroCompleto ) {
    this.#completo = numeroCompleto; // 👈
  }
  set #completo( valor ) { // 👈 Propriedade privada
    this.ddd = ( valor || '' ).substring( 0, 2 );
    this.numero = ( valor || '' ).substring( 2 );
  }
  get formatado() {
    return `(${this.ddd}) ${this.numero}`;
  }
}
const tel = new Telefone( '22988887777' );
console.log( tel.formatado ); // (22) 988887777
tel.#completo = '22977776666'; // ⚠️ Erro (privado)
```

atributos e métodos estáticos

alocação de estáticos

atributos e métodos **estáticos**

são alocados na memória

ao processar a declaração da classe

pela primeira vez.

atributos e métodos **estáticos**

pertencem à classe,

e podem ser acessados

sem necessidade de instanciar a classe.

exemplo 1 – atributo estático

```
class Impressora {
  static impressoes = 0; // 👈

  imprimir( algo ) {
    Impressora.impressoes++; // 👈 Via nome da classe
    console.log( algo );
  }
}

console.log( Impressora.impressoes ); // 0
const imp = new Impressora();
imp.imprimir( 'Olá' );
imp.imprimir( 'Mundo' );
const imp2 = new Impressora();
imp2.imprimir( '!' );
console.log( Impressora.impressoes ); // 3
```

exemplo 2 – métodos estáticos

```
class Calculo {
  static soma( x, y ) {
    return x + y;
  }
  static media( x, y ) {
    return Calculo.soma( x, y ) / 2;
  }
}

console.log( Calculo.soma( 10, 20 ) ); // 30
console.log( Calculo.media( 10, 20 ) ); // 15
```

exemplo 3 – propriedades estáticas

```
class Impressora {
    static #impressoes = 0; // 👈 Estática e privada

    static get impressoes() { return Impressora.#impressoes; } // 👈

    imprimir( algo ) {
        Impressora.#impressoes++; // 👈
        console.log( algo );
    }
}


console.log( Impressora.impressoes ); // 0
const a = new Impressora();
a.imprimir( 'Olá' );
console.log( Impressora.impressoes ); // 1

Impressora.#impressoes = 0; // ⚠️ Erro (privado)
Impressora.impressoes = 0; // ⚠️ Erro (propriedade apenas para obter valor)
```

desalocação de estáticos

atributos e métodos **estáticos**

permanecem na memória

até o fim da execução da aplicação.

Logo, use-os com moderação.

classes anônimas

exemplo 1 – expressão com atribuição

```
const Pessoa = class {
  constructor( nome ) {
    this.nome = nome;
  }
};

const p = new Pessoa( 'Ana' );
console.log( p.nome );

const PessoaJuridica = Pessoa; // Ref. com novo nome
const pj = new PessoaJuridica( 'Acme' );
console.log( pj.nome );
```

exemplo 2 – instanciação na declaração

```
const p = new class {
  constructor( nome ) {
    this.nome = nome;
  }
}( 'Ana' ); // 👈 Passa o argumento

console.log( p.nome ); // Ana
```

exemplo 3 – herança na declaração

```
class Documento {
  valido() { return false; }
}

const cpf = new class extends Documento {
  constructor( numero ) { super(); this.numero = numero; }

  valido() {
    return /^[0-9]{11}$/.test( this.numero );
  }
}( '11111111111' ); // 👈 Passa o argumento

console.log( cpf.valido() ); // true
```

referências usadas

MDN. **Referência JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference

fim

2025.09.29 – Nova versão completa, com ajustes no conteúdo, acréscimo de visibilidade privada e classes dinâmicas com extensão.
2020.01.20 – versão inicial.