POO em JavaScript

Programação Orientada a Objetos

Prof. Rone Ilídio

Objeto

- Variável formada por código e dados
- Toda variável é uma área de memória para armazenar dados e possui um nome
- O objeto, além de armazenar dados, possui funções, as quais serão chamadas de métodos
- Duas formas de se criar objetos:
  - Objeto Literal
  - Objeto instância de Classe

Objeto Literal

Objeto Literal

- Um objeto literal é uma estrutura de dados
- composta por atributos e métodos
  - Atributos: variáveis
  - Métodos: são funções que, normalmente, manipulam os dados dos atributos
- Em um objeto literal, os dados (valores dos atributos) são inseridos na criação
- Exemplo: objeto quadrado (próximo slide)

Objeto Literal

```
var retangulo = {
        base: 10,
        altura: 5,
        area: function(){
            return this.base * this.altura
        },
        perimetro: function(){
            return 2*this.base + 2*this.altura
        }
}
document.write('Base:'+retangulo.base)
document.write('<br>Altura:'+retangulo.altura)
document.write('<br>Área:'+retangulo.area())
document.write('<br>Perímetro:'+retangulo.perimetro())
```

Sintaxe para criação de objeto literal

```
var nomeDoObjeto = {
        atributo1: valor1,
        atributo3: valor3,
        ...
        metodo1 = function() {},
        metodo2 = function() {},
        ...
};
```

- Membro de um objeto é um atributo ou um método
- Vírgula (,) separa os membros
- Dois pontos (:) entre o nome do atributo e seu valor

O uso do this

- this é uma palavra reservada que quer dizer “o próprio objeto”
- É utilizado dentro do objeto para acessar os membros do próprio objeto

```
var pessoa1 = {
      nome: 'José',
      saudacao: function(){
            return 'Oi! Eu sou o ' + this.nome;
      }
};
var pessoa2 = {
     nome: 'Manuel',
     saudacao: function(){
            return 'Oi! Eu sou o ' + this.nome;
     }
 };
 document.write(pessoa1.saudacao()+’   '+pessoa2.saudacao())
```

Tipos de Atribuitos e Métodos

- Um objeto literal pode ter qualquer tipo de atributo e qualquer tipo de método
- Atributos e métodos são tratados da mesma forma que variáveis e funções, a diferença é que coloca-se o nome do objeto e um ponto (.) para acessar-los.
- Métodos podem ser criados das três formas, como as funções

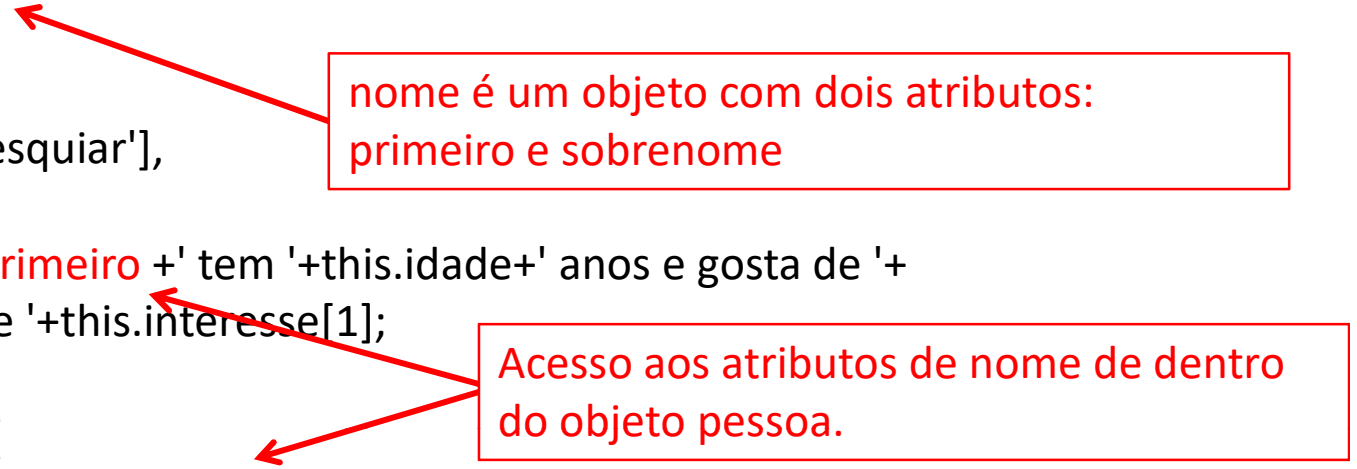
```
var pessoa = {
   nome: 'Jose',
   idade: 32,
   sexo: 'masculino',
   interesse: ['música','esquiar'],
   bio: function() {
    return this.nome +' tem '+this.idade+' anos e gosta de '+ this.interesse[0]+
              ' e '+this.interesse[1];
   },
   saudacao:  function() {
    return 'Oi! Eu sou o '+this.nome;
   }
 }
 document.write('Nome:' + pessoa.nome);
 document.write('<br>Interesses:' + pessoa.interesse[0])+ ‘ e ’+ pessoa.interesse[1]);
 document.write('<br>Saudação:' + pessoa.saudacao());
 document.write('<br>Biografia:' + pessoa.bio());
```

Vários tipos diferentes de dados: string, number e array.

Atributo Sendo Objeto

- Os atributos podem ser objetos
- O ponto (.) é utilizado para acessar
- No exemplo a seguir, nome é um objeto que atributo de pessoa. Nome possui dois atributos (primeiro e sobrenome)

```
var pessoa = {
    nome: {primeiro: 'Jose', sobrenome: 'Manuel'},
    idade: 32,
    sexo: 'masculino',
    interesse: ['música','esquiar'],
    bio: function() {
        return this.nome.primeiro +' tem '+this.idade+' anos e gosta de '+
        this.interesse[0]+' e '+this.interesse[1];
    },
    saudacao: function(){
        return 'Oi! Eu sou o '+this.nome.primeiro + ' ' + this.nome.sobrenome;
    }
}
document.write('Nome:' + pessoa.nome.primeiro);
document.write('<br>Sobrenome:' + pessoa.nome.sobrenome);
document.write('<br>Biografia:' + pessoa.bio());
document.write('<br>Saudação:' + pessoa.saudacao());
```



Criação de Membros em Tempo de Execução

- O JavaScript não é fortemente tipado
- Isso quer dizer que em tempo de execução:
  - Variáveis podem mudar de tipos
  - Objetos podem receber novos métodos e atributos

```
var pessoa = {
   nome: {primeiro: 'Jose', sobrenome: 'Manuel'},
   idade: 32,
   sexo: 'masculino',
   interesse: ['música','esquiar'],
   bio: function() {
      return this.nome.primeiro +' tem '+this.idade+' anos e gosta de '+ this.interesse[0]+
      ' e '+this.interesse[1];
   },
   saudacao: function(){
      return 'Oi! Eu sou o '+this.nome.primeiro + ' ' + this.nome.sobrenome;
   }
}
pessoa.olhos = 'castanhos';
pessoa.despedida = function () {
   alert('Tchau galera!');
};
document.write('Olhos:' + pessoa.olhos);
pessoa.despedida();
```

Criando um Objeto Literal a Partir de Outro

- O método Object.create(obj) cria um objeto a partir do objeto obj
- O novo objeto possuirá os mesmos métodos e os mesmos atributos, com seus respectivos valores
- O objeto criado é independente do primeiro

```
var pessoa = {
    nome : 'zé'
}
document.write('pessoa:'+pessoa.nome);
var gente = Object.create(pessoa);
document.write("<br>gente:"+gente.nome);
gente.nome='Mané'
document.write('<br>pessoa:'+pessoa.nome);
document.write("<br>gente:"+gente.nome);
```

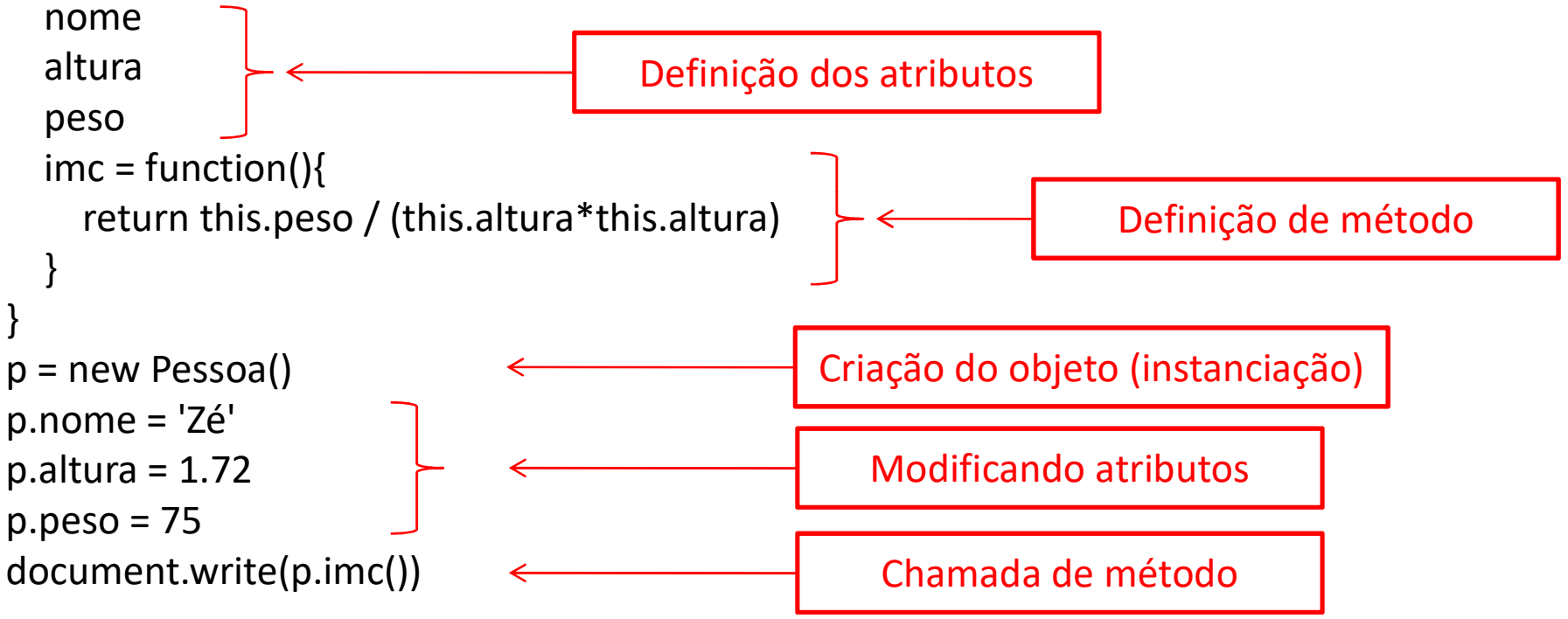
Classe

Classe

- Classes são tipos de dados definidos pelo programador
- A variável cujo tipo é uma classe é chamada de objeto
- A criação de um objeto é chamada de instanciação
  - Um objeto é uma instância de uma classe
- Uma classe é formada por
  - Atributos: variáveis
  - Métodos: funções que normalmente manipulam os atributos.
- Veja exemplo da classe Pessoa no próximo slide

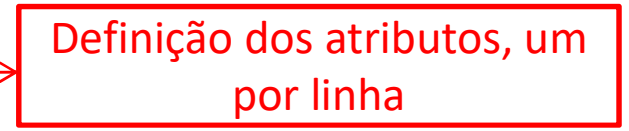
Classe

```
class Pessoa{
    nome
    altura
    peso
    imc = function(){
        return this.peso / (this.altura*this.altura)
    }
}
p = new Pessoa()
p.nome = 'Zé'
p.altura = 1.72
p.peso = 75
document.write(p.imc())
```



Sintaxe para Criação de Classe

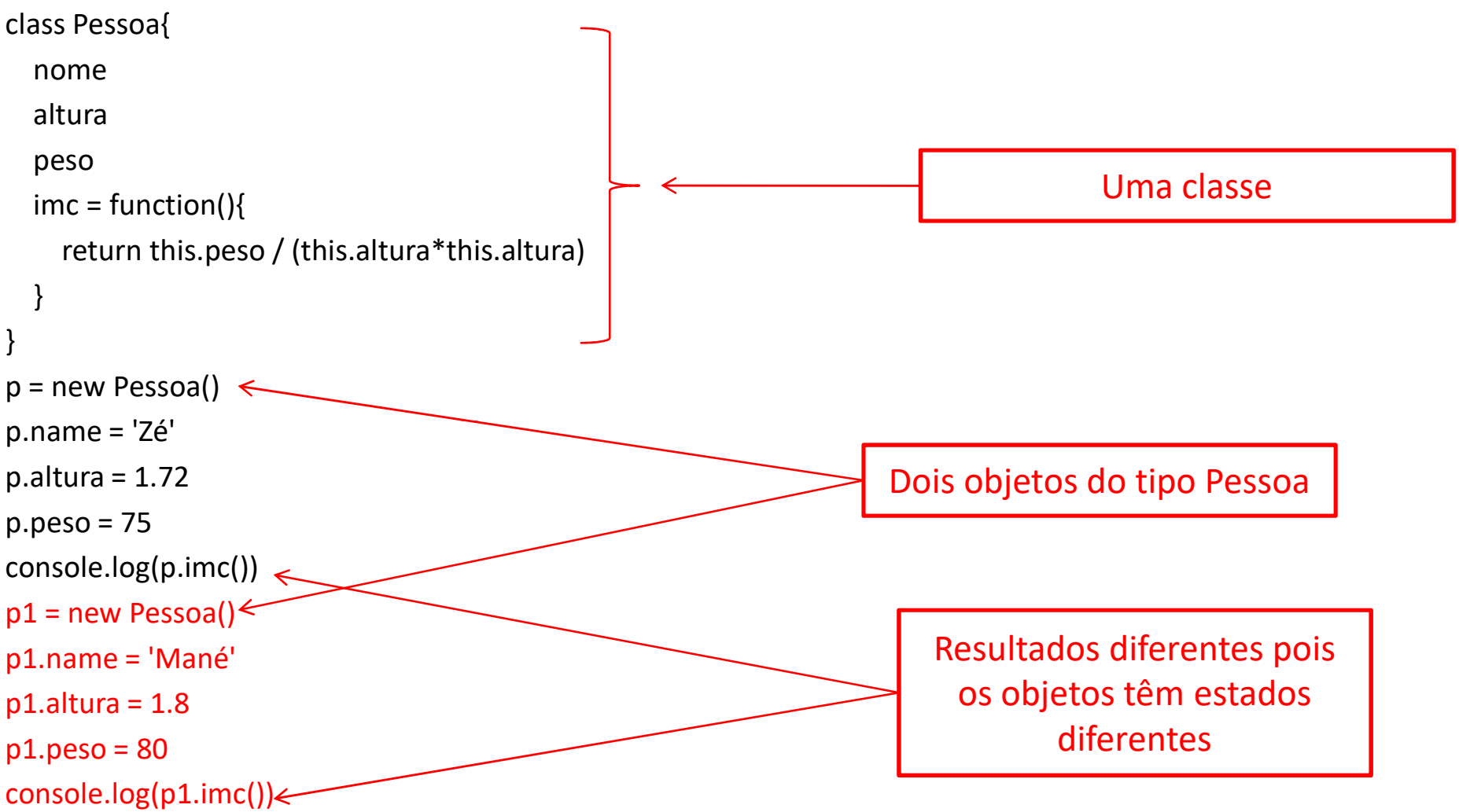
```
class NomeClasse{
    atributo1
    atributo2
    atributo3
    ...
    nomeMetodo = () => {
            ... Código do método ...
    }
    ... Definição de métodos, quantos forem necessários ...
}
```



Vários Objetos de uma Classe

- Pode-se instanciar de uma classe quantos objetos forem necessários
- Cada objeto será uma área de memória
  - A estrutura é a mesma
  - O estado dos objetos muda (estado corresponde aos valores dos atributos)
- No exemplo do próximo slide, foram instanciados os objetos p e p1, todos do tipo Pessoa.

```
class Pessoa{
   nome
   altura
   peso
   imc = function(){
      return this.peso / (this.altura*this.altura)
   }
}
p = new Pessoa()
p.name = 'Zé'
p.altura = 1.72
p.peso = 75
console.log(p.imc())
p1 = new Pessoa()
p1.name = 'Mané'
p1.altura = 1.8
p1.peso = 80
console.log(p1.imc())
```



Método Construtor

Método Construtor

- Método executado logo após a criação do objeto
- Definido pela palavra reservada *constructor*
- Utilizado para definir o estado inicial de objetos
- Estabelece os parâmetros para criação de objetos
- Se o construtor não for declarado o JavaScript considera um construtor vazio

```
class Pessoa{
        constructor (nome,idade){
                this.nome = nome;
                this.idade = idade;
        }
        saudacao(){
                return "Oi, sou o " + this.nome +
                        " e tenho "+this.idade+" anos.";
        }
}
var p = new Pessoa('Zé',18);
document.write(p.saudacao());
```

Definição do construtor que recebe dois parâmetros e cria dois atributos para a classe.

A criação do objeto deve respeitar os parâmetros definidos no construtor.

```
class Person{
      constructor(){}
          nome = "";
          idade = 0;
          saudacao(){
              return "Oi, sou o " + this.nome +
                          " e tenho "+this.idade+" anos.";
       }
}
var p = new Person();
p.nome = 'Zé';
p.idade = 18;
document.write(p.saudacao());
```

O construtor vazio pode ser declarado ou omitido

Neste caso, a chamada do construtor não tem parâmetros

Passagem Por Referência entre Objetos

Passagem Por Referência entre Objetos

- Quando um objeto recebe outro, ocorre uma passagem por referência → os dois objetos correspondem à mesma área de memória

Passagem Por Referência entre Objetos

```
class Person {
   constructor(){
      this.nome=String;
   }
}
var p = new Person();
p.nome = 'Zé'
var p2 = p;
p2.nome = 'José'
console.log(p.nome)
console.log(p2.nome)
```

Um objeto recebe o outro.
p e p2 ocupam a mesma área de memória. Alterando um também altera o outro.

Resultado

José

José

Criando um objeto a partir de um já existente

```
class Person {
   constructor(){
      this.nome=String;
   }
}
var p = new Person();
p.nome = 'Zé'
var p2 = Object.create(p);
p2.nome = 'José'
console.log(p.nome)
console.log(p2.nome)
```

Resultado:

Zé

José

Herança

Herança

Herança é a forma de utilização de software em que novas classes são criadas a partir de classes existentes, absorvendo seus atributos e comportamentos, e adicionando novos recursos que as novas classes exigem.

Herança

- Em outras palavras ...
- Cria-se uma classe que recebe (herda) os os membros que foram definidos em outra classe.
- O objeto dessa nova classe terá os membros das duas classes
- A classe que fornece os membros é chamada mãe
- A classe que recebe é chamada de filha

Herança

```
class Person{
    constructor (nome,idade){
      this.nome = nome;
      this.idade = idade;
    }
    saudacao(){
        return "Oi, sou o " + this.nome +
                    " e tenho "+this.idade+" anos.";
    }
}
```

Classe Mãe

```
class Staff extends Person{
    constructor(nome, idade, matricula){
        super(nome,idade);
        this.matricula=matricula;
    }
    apresentacao(){
        return "Oi, sou o " + this.nome +
                    " e tenho "+this.idade+" anos,"+
                    " matrícula:"+this.matricula;
    }
}
```

Classe Filha

Herança

- Atenção à class Staff
  - A palavra reservada extends define a herança
  - A chamada super(nome,idade) representa a chamada do método construtor da classe mãe
    - Como Person tem construtor, se torna obrigatório chamar super() na 1ª linha do construtor da classe filha
    - Se super não for chamado, aparece um erro
    - No navegador esse erro só aparece ao clicar F12

Herança

- Instanciação e utilização de objetos das classes Person e Staff:

```
var s = new Staff('Mané',10,12345);
document.write("<br>"+s.saudacao());
document.write("<br>"+s.apresentacao());
```

- O objeto s possui:
  - os atributos nome e idade → recebeu de Person
  - O método saudacao() → recebeu de Person
  - O átributo matrícula → definido em Staff
  - O método apresentacao() → definido em Staff

Sobrecarga de Método

Sobrecarga de Métodos

- Ocorre quanto as classes mãe e filha possuem um método com o mesmo nome
- Regra básica:
  - O método da filha é sempre chamado
- No exemplo a seguir
  - O método run() é declarado na classe mãe (Carro) e nas classes filhas (Mercedes e Honda).
  - Quando esse método é chamado a partir do objeto, é executado o run() que foi definido na classe filha

```
class Carro{
    nome
    constructor(nome){
        this.nome = nome
    }
    run (vel=0){
        console.log(this.nome + ' está em velocidade '+vel+' Km/h.')
    }
}
class Mercedes extends Carro{
    constructor(nome){
        super(nome)
    }
    run(vel = 150){
        console.log('Mercedes iniciou!')
        super.run(vel)
    }
}
```

Declaração do método run() na classe mãe.

Declaração do método run() na classe filha.

```
class Honda extends Carro{
    constructor(nome){
        super(nome)
    }
    run(vel= 100){
        console.log('Honda iniciou!')
        super.run(vel)
    }
}
```

Declaração do método run() na classe filha.

```
let objcarro = new Carro('Fusca')
let objmercedes = new Mercedes('Mercedes GTA')
let objhonda = new Honda('Honda Fit')
objcarro.run()
objmercedes.run()
objhonda.run()
```

O método run() da filha é chamado.

Funções como Objetos

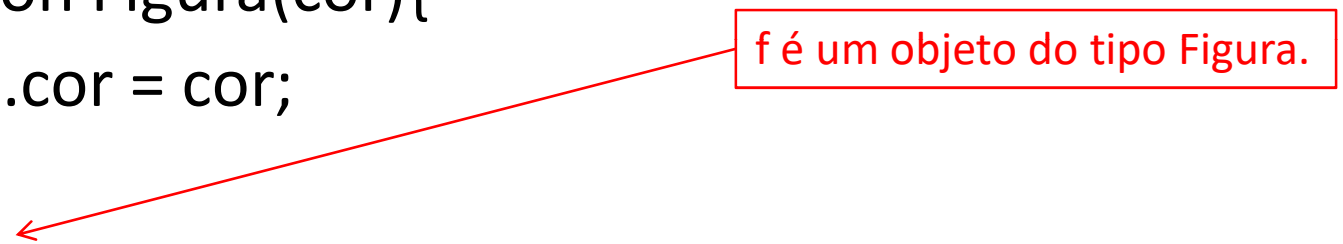
Funções como Objetos

- No JavaScript, uma função é tratada como um objeto
- Como visto anteriormente, toda função é filha de Function, que é filha de Object
- O comando new pode ser utilizado para criar objetos cujo tipo é uma função

Funções como Objetos

- f é um objeto cujo tipo é Figura, que é uma função

```
function Figura(cor){
    this.cor = cor;
}
f = new Figura('azul');
console.log(f.cor);
```



Encapsulamento

Encapsulamento

- Encapsulamento quer dizer encapsular ou esconder
- Define quais atributos e métodos estarão disponíveis para acesso em objetos e em classes filhas
- Utilizado para garantir que o acesso a um determinado recurso seja realizado de uma única forma
- Duas formas de encapsulamento:
  - Publico
  - Privado
- Obs: o Javascript não possui encapsulamento protegido (protected)

Encapsulamento

- Em uma classe, todos os membros são públicos por definição
- Um membro privado é definido com um # na frente de seu nome
- O membro privado só pode ser utilizado dentro da classe
- O membro público pode ser utilizado em qualquer lugar

- Obs: o # não está implementado em versões mais antigas do JS. O padrão é utilizar “_” no início do nome dos atributos para identificar atributos privados. Mas isso é só convenção, não está na sintaxe de JS antigo.

Encapsulamento

```
class Pessoa{
   #nome
   getNome = function(){
      return this.#nome
   }
   setNome = function(nome){
      this.#nome = nome
   }
}
p = new Pessoa();
p.setNome('Zé')
console.log(p.getNome())
```

O atributo nome é privado.

Ele só pode ser utilizado dentro da classe Pessoa.

Tente: console.log(p.#nome)
Isso leva a um erro, pois #nome é private

Encapsulamento

- O padrão utilizado em diversas linguagens de programação é proteger o atributo e criar métodos get e set para leitura e escrita em cada atributo.
- Atributos privados não são acessíveis nas classes filhas pois não são passados na herança
- No exemplo a seguir, o atributo cor é privado.
  - Pode ser utilizado dentro da classe Figura
  - Não pode ser utilizado dentro da classe Quadrado
  - Não pode ser utilizado no objeto

Encapsulamento

```
class Figura{
    #cor
    getCor = () => {
        return this.#cor
    }
    setCor = (cor) => {
        this.#cor = cor
    }
}
```

```
class Cubo extends Figura{
    #lado
    getLado = () => {
        return this.#lado
    }
    setLado = (lado) => {
        this.#lado = lado
    }
}
```

Dentro da classe Cubo não é possível acessar o atributo #cor

Encapsulamento

- Utilização das classes no corpo principal

```
c = new Cubo()
c.setCor('Vermelho')
c.setLado(10)
console.log('Cor:'+c.getCor())
console.log('Lado:'+c.getLado())
```

O objeto c tem acesso aos métodos setCor() e setLado(), pois são públicos.

Ao final, digite c. e espere o autocomplemento. Verifique que somente os métodos públicos estarão disponíveis.

UML

Unified Modeling Language

UML

- Linguagem para modelagem de sistemas computacionais
- Formada por vários diagramas padronizados
- Utilizaremos somente o Diagrama de Classes
- Define uma representação gráfica para:
    - Classes
    - Relacionamentos entre classes
    - Atributos
    - Encapsulamento

UML

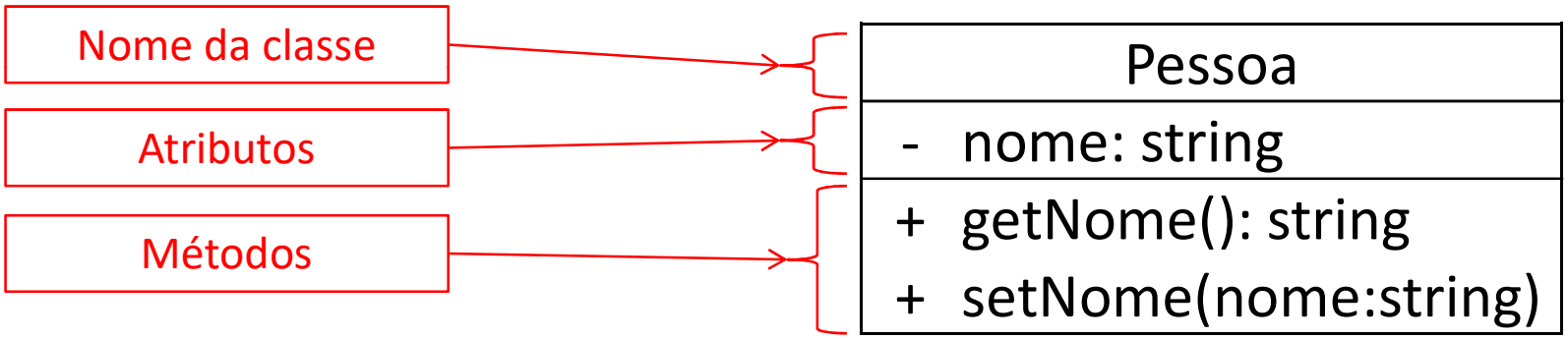
```
class Pessoa {
    #nome
    getNome = () => {
        return this.#nome
    }
    setNome = (nome) => {
        this.#nome = nome
    }
}
```

Pessoa
- nome: string
+ getNome(): string + setNome(nome:string)

- → privado

+ → público

UML


Nome da classe
Atributos
Métodos
Pessoa
- nome: string
+ getNome(): string
+ setNome(nome:string)

UML

```
class Cliente extends Pessoa{
    #cpf
    getCpf = () => {
        return this.#cpf
    }
    setCpf = (cpf) => {
        this.#cpf = cpf
    }
}
```