UNINOVE

Programação Orientada a Objetos

INTRODUÇÃO A ORIENTAÇÃO A OBJETOS EM JAVASCRIPT

AUTOR(A): PROF. DANIEL FERREIRA DE BARROS JUNIOR

JavaScript – Orientação a Objetos

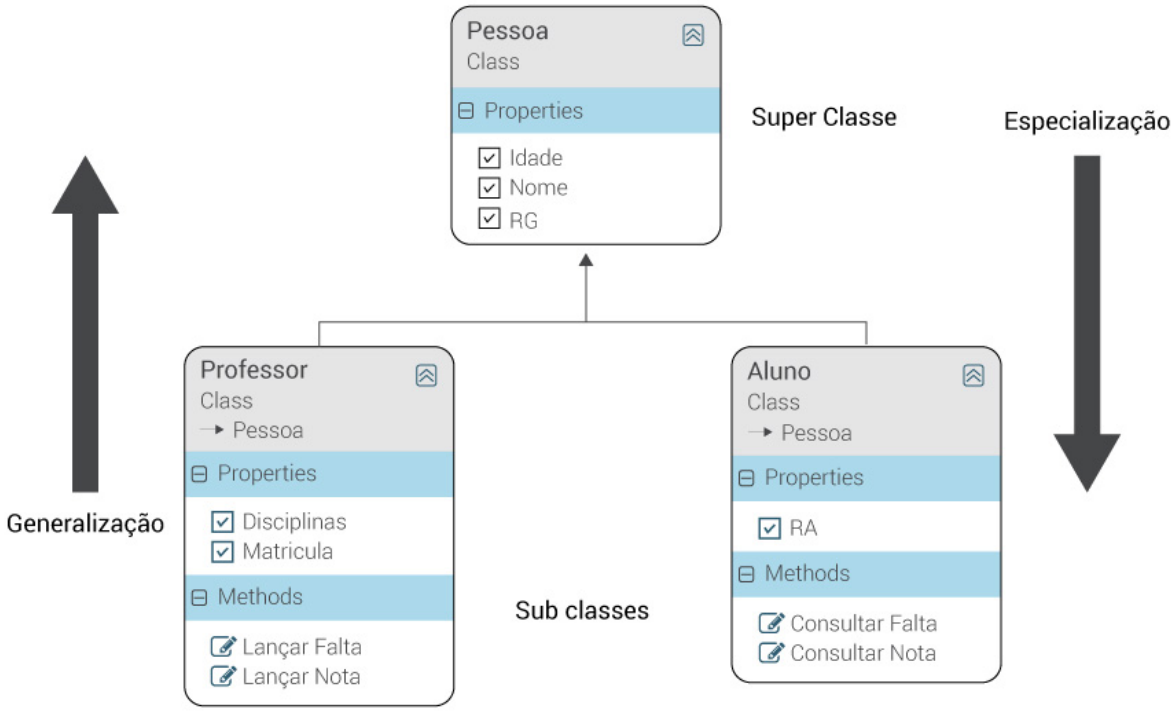
Neste último tópico vamos apresentar novos conceitos da orientação a objetos. Novamente, não esperamos ou pretendemos em um único tópico abordar ou apresentar todos os elementos da orientação a objetos.
Antes de apresentar o exemplo final, vamos listar resumidamente alguns conceitos.

Herança

O conceito de herança é muito similar ao exemplo biológico, onde um pai ou uma mãe podem possuir descendentes, seus filhos, ou seja, os filhos herdam de seus pais algumas das características genéticas.
Em orientação a objetos é muito semelhante, pois uma classe filho, ou subclasse, herda os atributos e métodos da classe pai, ou superclasse.
Em outras palavras, uma classe possui a habilidade de herdar características de outra classe, a isso damos o nome de Herança.
Veja este exemplo:

```
1.          // Herança
2.          // Pessoa => Aluno
3.          Aluno.prototype = new Pessoa.prototype.constructor();
4.          Aluno.prototype.constructor = Aluno;
5.          // Opção para criar a Herança = Aluno.prototype = Object.create(Pe
6.          Funcionario.prototype = new Pessoa.prototype.constructor();
7.          Funcionario.prototype.constructor = Funcionario;
```

Na imagem abaixo, podemos verificar uma superclasse, com seus dados mais genéricos, interligada com suas subclasses, mais específicas, em uma relação de herança entre classes "pai" e "filho".



Legenda: EXEMPLO DE HERANçA

Encapsulamento

Uma maneira de reunir e proteger os atributos e os métodos de uma classe. Uma vez isolados, o acesso ou modificação de dados só podem ser realizar pela própria classe, que disponibiliza métodos específicos para esta ação. Estes métodos normalmente são nomeados com métodos getters (leitura) e setters (alteração). Exemplo:

```
<script type="text/javascript">
   var Pessoa = function (nome){
           var idade;
           return{
                   setNome: function(newNome){
                           nome = newNome;
                   },
                   getNome: function(){
                           return nome;
                   },
                   setIdade: function(newIdade){
                           idade = newIdade;
                   },
                   getIdade: function(){
                           return idade;
                   }
           }
   }

   var aluno = Pessoa('Eduardo');
   document.write("<br>Nome do aluno: " + aluno.getNome());   // retorna E
   aluno.setNome('Joao');                                     // alterado
   aluno.setIdade(20);
   document.write("<br>Idade: " + aluno.getIdade());          // retorna 2
   document.write("<br>Nome Atualizado: " + aluno.getNome()); // retorna J

</script>
```

Abstração

A abstração é formada por um conjunto de heranças, onde uma classe abstrata pode apenas servir de modelo para outras classes, ou seja, um objeto não pode ser instanciado de uma classe abstrata.
A criação de classes abstratas serve para criar um maior grau de generalização entre classes.

Polimorfismo

Quando mais de uma classe utiliza um mesmo método ou atributo, podemos utilizar o polimorfismo para implementar esta funcionalidade.

Entre uma das vantagens é padronizar a nomenclatura dos métodos, com comportamentos diferentes e de acordo com a sua respectiva classe.

Exemplo:

```
1. // Método exibir
2.          Pessoa.prototype.exibirDados = function(){
3.              return 'Nome = ' + this.getNome() + ', idade = ' + this.getIda
4.          }
5.
6.          // Polimorfismo do método exibirDados
7.          Aluno.prototype.exibirDados = function(){
8.              return Pessoa.prototype.exibirDados.call(this) +
9.              ', RA = ' + this.ra + ', curso = ' + this.curso + '.';
10.         }
11.
12.         // Polimorfismo do método exibirDados
13.         Funcionario.prototype.exibirDados = function(){
14.             return Pessoa.prototype.exibirDados.call(this) +
15.             ', Matricula = ' + this.matricula + ', Setor = ' + this.setor
16.         }
```

No exemplo abaixo veremos a implementação de algumas destas caracteristicas da orientação a objetos.

Veja o exemplo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Tópico 20</title>
    <meta charset="UTF-8">
</head>
<body>
    <script>
        // Classe pai (Superclasse)
        function Pessoa(a, b){
            this.nome = a;
            this.idade = b;
        }

        // Classe filho (Subclasse)
        function Aluno(a, b, c, d){
            Pessoa.call(this, a, b);
            this.ra = c;
            this.curso = d;
        }

        // Classe filho (Subclasse)
        function Funcionario(a, b, c, d){
            Pessoa.call(this, a, b);
            this.matricula = c;
            this.setor = d;
        }

        // Herança
        // Pessoa => Aluno
        Aluno.prototype = new Pessoa.prototype.constructor();
        Aluno.prototype.constructor = Aluno;
        // Opção para criar a Herança = Aluno.prototype = Object.create(Pe
        Funcionario.prototype = new Pessoa.prototype.constructor();
        Funcionario.prototype.constructor = Funcionario;


        // Métodos
        // Métodos acessores
        Pessoa.prototype.getNome = function(){
```

```
41.             return this.nome;
42.         }
43.         Pessoa.prototype.getIdade = function(){
44.             return this.idade;
45.         }
46.
47.         // Métodos modificadores
48.         Pessoa.prototype.setNome = function(x){
49.             this.nome = x;
50.         }
51.         Pessoa.prototype.setIdade = function(x){
52.             this.idade = x;
53.         }
54.
55.         // Método exibir
56.         Pessoa.prototype.exibirDados = function(){
57.             return 'Nome = ' + this.getNome() + ', idade = ' + this.getIda
58.         }
59.
60.         // Polimorfismo do método exibirDados
61.         Aluno.prototype.exibirDados = function(){
62.             return Pessoa.prototype.exibirDados.call(this) +
63.             ', RA = ' + this.ra + ', curso = ' + this.curso + '.';
64.         }
65.
66.         // Polimorfismo do método exibirDados
67.         Funcionario.prototype.exibirDados = function(){
68.             return Pessoa.prototype.exibirDados.call(this) +
69.             ', Matricula = ' + this.matricula + ', Setor = ' + this.setor
70.         }
71.
72.         // Instanciar objetos
73.         var p1 = new Pessoa('Joao da Silva', 20);
74.         var p2 = new Aluno('Pedro', 19, '123456', 'Ciênca da Computação');
75.         var p3 = new Aluno('Maria da Graça', 34, '456123', 'Tec. em Siste
76.         var p4 = new Funcionario('Gustavo Lira', 26, '123', 'Secretaria');
77.         var p5 = new Funcionario('Roberto Kall', 60, '203', 'TI');
78.
79.         // Exibir
80.         console.log(p1.exibirDados());
81.         console.log(p2.exibirDados());
```

```
82.         console.log(p3.exibirDados());
83.         console.log(p4.exibirDados());
84.         console.log(p5.exibirDados());
85.
86.         // Alterações
87.         p1.setIdade(21);
88.         p2.setNome('Pedro Antonio');
89.         p5.setIdade(45);
90.
91.         // Array de objetos
92.         var universidade = [p1, p2, p3, p4, p5];
93.
94.         // Reexibir em loop
95.         console.log('\n***********************************');
96.         for(cadaPessoa of universidade){
97.             console.log(cadaPessoa.exibirDados());
98.         }
99.
00.         // Procurar o aluno mais velho da universidade
01.         var maiorIdade = 0; var cadaPessoa;
02.         for(cadaPessoa of universidade){
03.             if( (cadaPessoa.getIdade() > maiorIdade) && (String(cadaPessoa
04.                 maiorIdade = cadaPessoa.getIdade();
05.                 var idPessoa = cadaPessoa;
06.             }
07.         }
08.
09.         console.log('\nO aluno mais velho é o ' + idPessoa.getNome() + ',
10.
11.    </script>
12. </body>
13. </html>
```

A imagem a seguir apresenta o resultado no navegador para o código apresentado acima:

```
Nome = Joao da Silva, idade = 20 anos                                                    aula20_a.html:80
Nome = Pedro, idade = 19 anos, RA = 123456, curso = Ciênca da Computação.                aula20_a.html:81
Nome = Maria da Graça, idade = 34 anos, RA = 456123, curso = Tec. em Sistemas para Internet.  aula20_a.html:82
Nome = Gustavo Lira, idade = 26 anos, Matricula = 123, Setor = Secretaria.               aula20_a.html:83
Nome = Roberto Kall, idade = 60 anos, Matricula = 203, Setor = TI.                       aula20_a.html:84
                                                                                         aula20_a.html:95
**********************************
Nome = Joao da Silva, idade = 21 anos                                                    aula20_a.html:97
Nome = Pedro Antonio, idade = 19 anos, RA = 123456, curso = Ciênca da Computação.        aula20_a.html:97
Nome = Maria da Graça, idade = 34 anos, RA = 456123, curso = Tec. em Sistemas para Internet.  aula20_a.html:97
Nome = Gustavo Lira, idade = 26 anos, Matricula = 123, Setor = Secretaria.               aula20_a.html:97
Nome = Roberto Kall, idade = 45 anos, Matricula = 203, Setor = TI.                       aula20_a.html:97
                                                                                         aula20_a.html:109
O aluno mais velho é o Maria da Graça, tem 34 anos!
>
```

Assista os vídeos a seguir, nestes são realizados a implementação comentada do programa acima.
Caso necessário, assista os vídeos mais de uma vez.

> ## DICA:
>
> Para saber mais sobre orientação a objetos em JavaScritp, veja em:
> https://developer.mozilla.org/pt-PT/docs/Javascript_orientado_a_objetos
> (https://developer.mozilla.org/pt-PT/docs/Javascript_orientado_a_objetos)
> https://www.w3schools.com/js/js_object_prototypes.asp
> (https://www.w3schools.com/js/js_object_prototypes.asp)

Conclusão

Podemos considerar a herança, o encapsulamento, o polimorfismo e a abstração, como pilares do paradigma de programação orientada a objetos.
Apesar de ser um assunto extenso, sua aplicação em JavaScript é simples e pode ser implementada em poucos passos.
Mantenha a atenção e cuidado no projeto e codificação de seu script.

ATIVIDADE

Quanto a Herança em orientação a objetos, podemos afirmar:

(Escolha a alternativa correta)

A. A subclasse é uma classe inferior.

B. A subclasse é descendente da superclasse.

C. A Superclasse é a classe principal de todo programa.

D. A herança é a proteção e isolamento de atributos e métodos.

ATIVIDADE

Escolha a alternativa que complete a seguinte afirmação:

Os métodos getters são...

A. Métodos acessores, pois permitem acesso a atributos privados ou protegidos.

B. Métodos modificadores, pois permitem acesso a atributos privados ou protegidos.

C. Métodos destruidores, pois permitem acesso a atributos protegidos.

D. Métodos construtores, pois permitem acesso a atributos privados.

ATIVIDADE

Escolha a alternativa que corresponda ao código exibido a seguir:

```
Pessoa.prototype.setNome = function(x){
    this.nome = x;
}
```

A. Método acessor

B. Método construtor

C. Método modificador

D. Método destruidor

REFERÊNCIA

MORRISON, M. Use a cabeça JavaScript. 5º Ed. Rio de Janeiro: Alta Books, 2012. 606 p.

OLIVIERO. C. A. J. Faça um site JavaScript orientado por projeto. 6º ed. São Paulo: Érica, 2010. 266 p.

ZAKAS, Nicholas C. JavaScript de alto desempenho. 8º Ed. São Paulo: Novatec, 2010. 245 p.

TERUEL, Evandro Carlos. Programação orientada a objetos com JAVA sem mistérios. 1º ed. São Paulo, 2016: Universidade Nove de Julho - UNINOVE. 386 p.